TOPOLOGY CONTROL OF VOLUMETRIC DATA

A Thesis
Presented to
The Academic Faculty

by

James Vanderhyde

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Interactive Computing

Georgia Institute of Technology
August 2007

TOPOLOGY CONTROL OF VOLUMETRIC DATA

Approved by:

Dr. Andrzej Szymczak, Advisor
School of Interactive Computing
Georgia Institute of Technology

Dr. Konstantin Mischaikow
Department of Mathematics
Rutgers, The State University of New Jersey

Dr. Jarek Rossignac
School of Interactive Computing
Georgia Institute of Technology

Dr. Allen Tannenbaum
School of Biomedical Engineering
Georgia Institute of Technology

Dr. Greg Turk
School of Interactive Computing
Georgia Institute of Technology

Date Approved: 5 July 2007

ACKNOWLEDGEMENTS

I wish to thank my advisor, Andrzej Szymczak, for our relationship throughout my career at Georgia Tech. I am grateful for patience when I was slow in understanding or implementing a new idea and for treating my ideas with gentleness and encouragement.

I also wish to thank my other committee members from the College of Computing, Greg Turk and Jarek Rossignac, for all their interesting ideas and fun and exciting class projects. Finally, I wish to thank my external committee members, Konstantin Mischaikow and Allen Tannenbaum, for help with the theory and applications of my research.

I particularly want to thank all the other professors who have helped me over the years with projects or advice: Amy Bruckman, Irfan Essa, Dick Lipton, Dana Randall, and H. Venkateswaran from Georgia Tech; Eric Torng, Charles Ofria, Abdol-Hossein Esfahanian, and Don Weinshank at Michigan State University; and from Hope College, Herb Dershem, Mike Jipping, and Tim Pennings.

Finally, I want to thank my wife, Mariam Mathew, for love and support, especially as I tried to get this thesis finished. Our relationship has only just begun.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

Three-dimensional scans and other volumetric data sources often result in representations that are more complex topologically than the original model. The extraneous critical points, handles, and components are called "topological noise." Many algorithms in computer graphics require simple topology in order to work optimally, including texture mapping, surface parameterization, flows on surfaces, and conformal mappings. The topological noise disrupts these procedures by requiring each small handle to be dealt with individually. Furthermore, topological descriptions of volumetric data are useful for visualization and data queries. One such description is the contour tree (or Reeb graph), which depicts when the isosurfaces split and merge as the isovalue changes. In the presence of topological noise, the contour tree can be too large to be useful. For these reasons, an important goal in computer graphics is simplification of the topology of volumetric data.

The key to this thesis is that the global topology of volumetric data sets is determined by local changes at individual points. Therefore, we march through the data one grid cell at a time, and for each cell, we use a local check to determine if the topology of an isosurface is changing. If so, we change the value of the cell so that the topology change is prevented. We use this technique to simplify a single isosurface in a volume, and we use the technique to simplify all the isosurfaces in a volume together. For a single isosurface, the result is an object that looks like it has been shrink-wrapped in plastic. The surface of the object is the same as the input except that all the handles are patched over; the output is topologically equivalent to a sphere. To simplify all the isosurfaces together, we use the local check to reduce the number of critical points in the scalar field. This is done by starting at the global

minimum and adding cells one at a time in order by scalar value. If a cell changes topology, it is a critical point. In that case we delay adding the voxel until adding it does not cause a topology change, or until a user-specified threshold on the amount of delay is reached. This way only the critical points that are the most important are retained in the output.

In this thesis we describe variations on the local topology check for use in different settings. We use the topology simplification procedure to extract a single component with controlled topology from an isosurface in volume data sets and partially-defined volume data sets. We also use it to remove critical points from three-dimensional volumes, as well as time-varying volumes. We have applied the technique to two-dimensional (plus time) data sets and three dimensional (plus time) data sets.

CHAPTER 1

INTRODUCTION

The main thrust of this thesis lies in the domains of computer graphics and visualization. The goal of computer science is to teach the computer to manipulate input in a certain way to produce the desired output. In the case of graphics, the input is often a mathematical description of a scene, and the output is a visually attractive image displaying the scene. In visualization, the input is any sort of data set, whether abstract like a file system structure or spatial like a computed tomography (CT) scan, and the output is an image that effectively conveys the data set to the user. This thesis relates specifically to representations, descriptions, and analysis of geometric data sets. Computational geometry deals with representing the geometry of objects and developing algorithms to process the representations. It is used in graphics and visualization because users relate easily to geometric descriptions. The related field of computational topology deals with representing and processing the topology of objects. The topology of an object describes the number of components and the number of handles (and other related features) that the object has. This thesis relates with this field, since it deals with the processing of the topology of discrete data sets and the construction of combinatorial representations of the topology of the data.

These topological representations are useful because they are a small description of large amounts of data. Our main interest in computational topology lies in creating these simple descriptions, and the key to developing them is dimensionality reduction. Viewing 3D and higher-dimensional spatial data sets is challenging because the number of data points increases quickly as the dimension increases. One way to reduce the dimension so that viewing is easier is to extract surfaces. The

result is a two-dimensional representation that a user may pan around and view from different angles. When this is not sufficient, the dimension may be reduced further by constructing the contour tree, a one-dimensional representation that depicts when the surfaces in the volume split and merge. The contour tree is useful for storing information related to individual components of surfaces to make it easier to find and extract interesting components. It is also useful for cleaning up the data and identifying important, stable features. This idea of dimensionality reduction is used in many fields for many purposes and is a well-proven technique for understanding and visualizing high-dimensional data.

Three-dimensional volume data sets have become a popular representation of geometric data in recent years, due to advances in 3D medical imaging and other scanning techniques as well as rapid increases in available computer memory. Volume data sets consist of a scalar function defined at several sample points in a volume. For example, CT scans record a measure of tissue density at each point in a grid. A typical method of visualizing volumetric data is extraction of isosurfaces. An isosurface (or level set) is the set of points in the volume with value equal to a given value, the isovalue. Typically some sort of interpolation over the full volume is assumed since data only exist at discrete samples. In a three-dimensional volume, a level set is two dimensional and therefore easier to view. The benefits of extracting an isosurface include all the processing that can be done on 2D polygonal meshes to enhance visualization, such as texture mapping and simplification. However, many algorithms for these purposes get bogged down in details in the mesh that are not important to visualization such as small handles and holes. We call these details *topological noise* because they artificially increase the genus of the model, often to a great extent, and they typically appear due to inaccuracies in scanning procedures. Therefore, methods are needed to simplify the data set before we try to visualize it.

Even when the topological noise in the isosurfaces is reduced, when the volume

data set has a time dimension as well as two or three spatial dimensions, visualization is even more challenging. Furthermore, simple extraction of isosurfaces makes it difficult to see exactly how the isosurfaces change as the isovalue changes—how components (contours) split and merge or change genus. The contour tree is created by identifying each component of each isosurface to a point. Following a path in the tree corresponds to tracking a contour as the isovalue changes. The contour tree provides users with a simple one-dimensional object to manipulate when viewing the volume data. For example, an interface can be built that allows users click on points in the contour tree and see the corresponding contours extracted from the volume [15].

However, even the one-dimensional contour tree can quickly become difficult to manage when dealing with complex data sets. The structure of the contour tree is tied to the critical points of the underlying function (the maxima, minima, and saddle points), so one way to simplify the contour tree is to simplify the volume itself. In fact, reducing the number of critical points in the data will simplify the topology of the isosurfaces and therefore the structure of the contour tree. Once the data set is simplified, the contour tree may more easily be used to process, query, and visualize the volume.

There are many uses for volume data and topological processing in particular. Sanjay Rana [74] compiles methods of topological analysis applied most specifically to geographic information systems, including two-dimensional topographic data and urban population analysis. Scientific simulation and visualization have huge potential for topological analysis of volumes. For example, Pascucci and Cole-McLaughlin [71] show how a contour tree can be used to examine the structure of a methane molecule. Also, Sohn and Bajaj [81] show how contour trees can be used to view hemoglobin molecules carrying oxygen. Another source of volume data is medical imaging techniques such as MRI and CT. The time it takes for specialists to analyze these data sets can be dramatically reduced with the aid of topological processing.

For example, Szymczak et al. [86] use a topological approach to segment blood vessels from CT scans. Specialists can use the result to view the overall structure of the blood vessels rather than viewing the data slice by slice.

1.1 Problem Statement

Therefore, what is required is a topological representation and a topology simplification algorithm that simplify processing, querying, and visualization of volume data. The representation should be small for data sets of any dimensionality, including time-dependent data. The algorithm should be fast, simple, and easy to implement. It should also leave intact the regions of the data set not associated with the simplification. Finally, the algorithm should allow for extraction of watertight isosurfaces with important topological features.

1.2 Contributions

We reduce or eliminate the topological noise in a single isosurface by processing the surface in a 3D grid and marching through the grid cells one at a time. It is particularly appropriate for models that come from grid data such as alignment of laser range scans [62] or medical scans. The key to the project is that the global topology of a data set is determined by local changes at individual points. Therefore, we process the data cell by cell, and for each cell we use a local check to determine if the topology of the surface is changing. The result of this procedure is an object that looks like it has been shrink-wrapped in plastic. The surface of the object is the same as the input except that all the handles are patched over; the output is topologically equivalent to a sphere (see Figure 1).

Since volume data sets can be too large to fit comfortably in core memory, we implemented a procedure that operates directly on an octree representation of one isosurface the volume. Some volume data sets contain unknown regions, but this is not a problem for our technique because it automatically constructs a well-defined

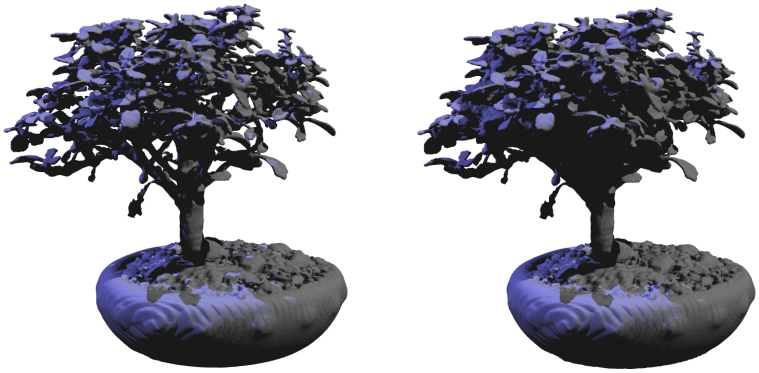


Figure 1: The isosurface extracted from a bonsai tree model. Left is extracted from the input (genus 463), and right is extracted from the output of our algorithm (genus 0).

sense of inside and outside for every voxel, so the extracted mesh will not only have controlled topology but will also be watertight.

We next apply the technique of the local check to topology simplification of scalar fields defined on a 3D grid. In this setting, topological noise consists of extraneous critical points in the scalar field. This method also processes cells one by one and determines whether the isosurfaces change topology. If so, the procedure delays the change in topology for as long as possible while other cells are processed. This way only the critical points that are in some sense the most important are retained. The result has fewer critical points and is guaranteed to be within a user-defined threshold of the input data. We also discuss a multiple-pass variant of our algorithm that decreases the number of voxels having different values in the input and output volumes.

For tracking of contours in volume data, we use a representation based on the contour tree, the subdomain-aware contour tree for time-dependent data described by Szymczak [85]. Contours are tracked in 2D and 3D over time by examining the relationship of the contour trees of individual slices with the contour trees of subsets of the whole volume. We use the topology simplification techniques with a special local topology check to clean up the input data before constructing the contour trees.

This way the description of the data is much smaller, easier to manipulate, and more useful for visualization.

Our topological representation is based on the contour tree, and the size of this is controlled by the amount of simplification performed by our algorithm. Our technique edits the volume directly and only edits the volume to remove critical points, so the rest of the volume is left untouched. Our algorithm does not require any data structures more complex than a priority queue and the asymptotic running time is bounded by the time spent sorting the input data, so it is fast and easy to implement. Data sets of any dimension are handled by simple variation of one part of the algorithm, the local check. Our algorithm constructs isosurfaces after cleaning up the topological noise. Therefore the techniques described in this thesis fulfill all the requirements for the needed topological representation and topology simplification algorithm.

1.3 Thesis overview

This thesis describes each of our contributions in its own chapter. Before the contributions, Chapter 2 discusses related work in the fields of topology simplification, topological representations, and processing of time-varying data. Chapter 3 introduces definitions and theorems needed to understand the topology of scalar functions defined on a discrete grid. It also explains the local check for topology change and defines the contour tree precisely.

Chapter 4 describes the topology simplification algorithm in detail. It is split into two sections. The first explains the algorithm for simplifying a given isosurface in a volume, and the second explains the algorithm for simplifying the whole volume. Chapter 5 is a brief aside on what our algorithm can do with partially-defined volumes.

Chapter 6 describes topology simplification and contour tracking in parameter-dependent data. We show an application of contour tracking in a human chest CT

scan to extract airways. Also we demonstrate the effectiveness of the topology simplification algorithm on time-varying 2D and 3D data by examining the size of the contour tree representation before and after simplification.

Finally, Chapter 7 is the conclusion. It includes a review of the contributions of the thesis, an explanation of some challenging examples, and a discussion of ideas for future work based on this thesis.

CHAPTER 2

RELATED WORK

2.1 Topology simplification

Much work has been done on topology simplification of volumetric data. The following is a sampling of uses and techniques.

2.1.1 Simplification of isosurface topology

Volume data sets are often used as an implicit representation of the surface of an object. They have many advantages over explicit representations (such as triangle meshes). However, sometimes a mesh is desired, especially for visualization purposes. In this case an isosurface is extracted from the volume, but the surface may have complex topology due to noise.

To extract an isosurface, van Overveld et al. [90, 10] use the idea of collapsing a membrane from outside the object in a technique they call shrinkwrapping. They begin with a topologically-trivial triangle mesh and guide it toward the isosurface using the volume data. The mesh is edited along the way to get around topological obstacles. Editing the mesh is quite complex and therefore in contrast we simplify the topology in voxel space before extracting a triangle mesh.

Guskov and Wood [43] also simplify the topology of meshes in the mesh domain. For "the advantages of efficiency and robustness," Wood et al. also developed a volumetric technique [93, 94] that simplifies the topology of an isosurface of a volume data set. They identify the topological noise in the volume data by constructing a Reeb graph for the entire volume and then identifying cycles in the Reeb graph. Handles smaller than a user-defined threshold are then patched or cut. This technique has very nice control over whether handles are patched or cut, but it is asymptotically

slower than our technique because whenever a change is made, the Reeb graph must be recomputed over the region of the change. Our technique is different in that we do not explicitly find all the noise handles but we patch or cut them all in one pass.

Bischoff and Kobbelt [8] explain how to preserve topology during "fast marching" methods such as ours. Their technique is interesting in that they represent topology changes as lower-dimensional faces and extract topologically-correct isosurfaces with subvoxel accuracy.

2.1.2 Simplification of volume geometry with topology control

A different application that uses techniques related to ours is simplification of the geometry of volumetric data. Since volumetric representations are amenable to local topological analysis, much work has been done on controlling the topology during geometry simplification.

Chiang and Lu [17] present a method for preserving topology during geometric simplification of a tetrahedral mesh. They simplify the mesh using edge collapses and use a contour tree to identify regions where edges can be collapsed without changing the topology.

Gerstner and Pajarola [41, 70] present a method for multiresolution isosurface extraction with control over topology of the resulting surface. Their goal is to retain the important topological properties of the isosurface at all resolution levels in order to provide a mesh at a resolution specified by the user. For multiresolution extraction they use an octree-like tetrahedral mesh bisection structure on a regular grid. They compute a persistence-like measure of the importance of critical points in the volume and preserve all important critical points. We use some techniques related to theirs, such as look-up tables for the local checks for criticality. Their criticality test is based on the same principle as our local topology checks. We also use a measure of persistence to gauge importance of critical points. Overall, their technique and goal

are similar to ours. Since we use only cubes and not tetrahedra to analyze the data, we do not have the added complication of tetrahedral decomposition. However, we make no attempt to extract isosurfaces at varying resolution.

2.1.3 Simplification of volume topology

This thesis is focused on simplifying the topology of isosurfaces and volumes while retaining the original geometry in regions not associated with the simplification. An early approach that our work and others are based on was developed by Aktouf et al. [1, 2]. Their goal was to patch handles in 3D binary data by removing voxels one by one using a local topology check. They also described how using various distance transforms to control the order of voxel removals controls the shape of the patches. The distance transform was also used to allow the most important handles to be opened.

Han et al. [46, 47, 48] build on the techniques of Aktouf et al. to preserve the topology of the level sets of a deformable model. This is more general since the volume data set is not binary, and they allow the handles to be either all broken or all patched. They describe the algorithm in terms of deformable models, but it is similar to our techniques for isosurface topology control.

Edelsbrunner et al. [27, 30, 20] define the idea of persistence, a measure of the importance of topological features in volume data. They also develop the Morse-Smale complex [25, 26, 12] as a data structure to describe the topology of the data. We prefer the contour tree (Reeb graph) for topological analysis because of its simplicity. Gyulassy et al. [44, 45] use the Morse-Smale complex for topology simplification. Their algorithm simplifies the data, but the Morse-Smale complex tracks all the homology in the data and therefore requires a complex and somewhat slow algorithm. Furthermore increasing the dimensionality of the data requires much more work using their technique.

An important reason to simplify the topology of a volume data set is for visualization using isosurfaces. Carr et al. [15] describe one approach to this. They construct a contour tree representing the volume and then prune edges of the contour tree and flatten regions of the volume to reflect the changes in the tree. The result is similar to our method of volume topology simplification, since both methods effectively cancel critical point pairs. Pascucci et al. [72] also use contour trees for isosurface extraction. They prioritize edges during the construction of the contour tree in order to construct a multiresolution visualization of the volume and the tree. Though these techniques are similar to ours, the respective authors also give no indication how to extend the techniques to time-dependent data.

Ju et al. [55] demonstrate a very interesting new combination of topology repair and user interaction, similar to what we envision as future work for our techniques.

2.2 Contour trees

Contour trees and Reeb graphs have long been used as one-dimensional representations of higher-dimensional objects. A contour tree is simply a Reeb graph on a simply-connected domain. When the domain is an arbitrary 2-manifold, typically Reeb graphs are used to analyze the shape of the manifold. When the domain is a volume, the domain is typically simply connected and so contour trees are sufficient.

2.2.1 Analysis of surfaces

Reeb graphs of surfaces are important for visualization and analysis of models. Shinagawa and Kunii [80] explain how to construct a Reeb graph from cross sections of complex 2-manifolds, such as the spiral structure of the human cochlea (inner ear). Cole-McLaughlin et al. [21] build on this and other work by discussing properties of loops in Reeb graphs of 2-manifolds. The loops are important because they correspond to topological features in the surfaces.

Hilaga et al. [50] use Reeb graphs for shape matching of triangle meshes. They

construct a multiresolution Reeb graph so that structures of varying size can be matched. Since the Reeb graph is a topological description of the object, differences in orientation and scale can easily be ignored.

2.2.2 Isosurface extraction

The earliest most popular use for contour trees was seed sets for extracting isosurfaces [89] from volume data. Isosurface visualization remains the predominant motivation for construction of contour trees.

The basic algorithm for computing the contour tree that we use is described by Carr [14, 13]. It involves the construction of two intermediary trees, the split tree and join tree. These constructions process the data by sweeping in order of sample value. Since we process the data in this order for topology simplification, Carr's method is a good fit for us. A merging algorithm constructs the complete contour tree from the split and join trees. The split and join trees are very nice structures in their own right and are useful for topological analysis of volume data.

Pascucci et al. [71, 73] describe a divide-and-conquer approach to computing the split and join trees. Their technique can be used for efficient parallel computation of the contour tree. We use it to create intermediate results for contour trees of subdomains of the volume.

Chiang et al. [18] give a theoretically faster, output-sensitive algorithm based on Carr' technique. They sort only the critical points in the data instead of the entire data set and follow monotone paths in the volume to construct the trees. This approach can be useful but does not seem necessary in our work.

Besides isosurface visualization, contour trees are commonly used for topological analysis of volumes. Some applications are decomposition of volumes [88] and volume matching [98].

2.2.3 Related structures

Some other topological data structures have also been proposed. Edelsbrunner et al. prefer Morse-Smale complexes [25, 12]. Rana has edited a book [74] that describes contour trees as well as what Rana prefers, surface networks. Surface networks and Morse-Smale complexes are closely related. We use contour trees for topological analysis because Morse-Smale complexes grow in complexity as the dimension of the data set increases, but contour trees remain one-dimensional.

2.3 Time-varying data

Visualizing time-varying volume data is a hot topic. Ma [65] surveys current applications and strategies for visualization.

Our data representation comes from Szymczak [85]. It consists of a contour tree of each (2D or 3D) time slice of the data set and a contour tree of each "thick slice," two adjacent slices together considered as one space. This is combined with maps from the slice (the "subdomain") contour trees into the thick slice contour trees. These maps are induced from the inclusion maps from the slices into the thick slices. The contour trees of the thick slices can be combined to form as much of the full-space contour tree as desired using the recursive approach described by Pascucci [71].

The construction of the contour trees of the thick slices depends on a definition of the interpolation used on the values between the slices. Often linear interpolation is used, but as shown in Figure 1 of [85] it can cause artifacts in the interslice topology. Sohn and Bajaj [81] suggest a method of checking the amount of overlap between the objects bounded by contours in adjacent time slices to avoid artifacts. However, we believe that the method of interpolation does not matter much, in the sense that the resulting topological analyses are not very different.

Edelsbrunner et al. [28] propose a time-varying Reeb graph for isosurface extraction of time-varying volume data. They represent the connection of slices over time

by connecting critical points in the contour trees in adjacent slices with segments. They explain 12 cases that describe what can happen between slices. We represent the connection of slices over time using the inclusion-induced map of the slice contour trees into the thick slice contour trees. We have not explored any deeper connection between these two representations.

CHAPTER 3

TOPOLOGY OF CUBICAL SCALAR FIELDS

The explorations in this thesis concerning the analysis and repair of topology are primarily in the area of cubical scalar fields. Our use of the word *cubical* in this thesis is derived from Kaczinski, et al. [57].

Definition A *cubical scalar field* is a function h that assigns a scalar value to the center of each d-dimensional cell of a regular rectangular lattice of R^d.

The dimension d can be any positive integer. In this thesis we deal mainly with two, three, and four dimensions. See Figure 2 for a 2D example.

Definition Each d-dimensional cell of the lattice of the cubical scalar field h is called a *voxel*. Note that it is an axis-aligned parallelepiped in R^d. The *value* of a voxel is the scalar value that h assigns to the center of the voxel.

In order to define the topology of a cubical scalar field, we need a well-defined sense of connectivity of the voxels. We often describe things in three dimensions when the other dimensions are analogous.

Definition Two voxels are *vertex connected* if they share a facet, an edge, or a vertex (a cell of any dimension lower than the dimension of the voxel). In three dimensions vertex connectivity is called *26-connectivity*, since one voxel has $3^3 - 1 = 26$ total neighbors. Note that in two dimensions there are $3^2 - 1 = 8$ vertex neighbors, and in d dimensions $3^d - 1$ vertex neighbors.

Definition Two voxels are *face connected* if they share a facet (a cell one dimension lower than the dimension of the voxel). In 3D face connectivity is also called

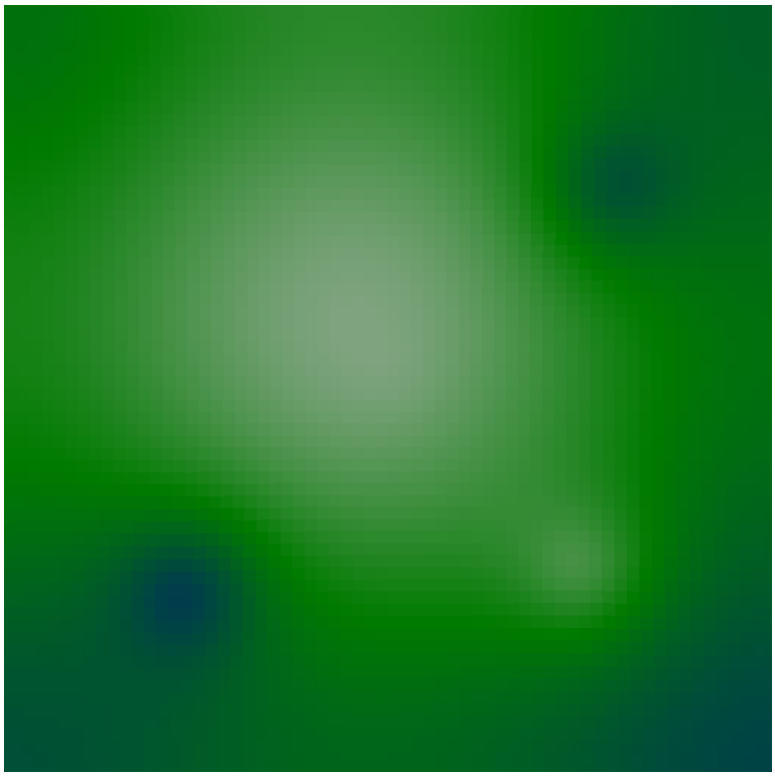


Figure 2: A cubical scalar field of dimension 2. Color corresponds to scalar values—white is high and blue is low.

6-connectivity, since every voxel has 6 neighbors that share a face (facet). In d dimensions a voxel has $2d$ face neighbors.

Note that edge-connectivity is also possible, but we do not consider it in this thesis.

Definition A *voxel set* is a pair $\langle S, c \rangle$, where S is a set of voxels and c is a given notion of connectivity represented by the number of neighbors a voxel has (either $3^d - 1$ for vertex or $2d$ for face connectivity). We often refer to S as the voxel set itself when the connectivity is clear from context, and for ease of discussion we use $c = 26$ or $c = 6$ by analogy from three dimensions, even when the dimension is not 3.

Definition The *realization* of a voxel set $\langle S, 26 \rangle$ in R^d is formed by treating each voxel as a closed subset of R^d (the voxel includes its boundary) and taking the union of the voxels in R^d. The realization is denoted $|S|_{26}$. The realization of a face-connected voxel set $\langle S, 6 \rangle$ is defined as the complement in R^d of the realization of the

complement of S treated as a vertex-connected set. Concisely, $|S|_6 = \overline{|\overline{S}|_{26}}$.

The realization of a finite voxel set is a "cubical set" as defined by Kaczinski, Mischaikow, and Mrozek in their book *Computational Homology* [57]. See that text for more information on cubical sets.

Definition The *complement* of a voxel set $\langle S, c\rangle$ is the set of all voxels not in S with the complementary notion of connectivity.

In other words, in 3D, $\overline{\langle S, 26\rangle} = \langle \overline{S}, 6\rangle$. Note that this definition of complement means in particular that $\overline{\overline{\langle S, c\rangle}} = \langle S, c\rangle$.

Definition A voxel set is *bounded* if it contains a finite number of voxels. Otherwise it is *unbounded*. The complement of a bounded voxel set is unbounded. Though an unbounded voxel set may have an unbounded complement, we do not consider any such sets in this thesis. Every voxel set either is bounded or has bounded complement.

3.1 Local Topology Check

We would like to analyze the topological invariants of voxel sets. These are values assigned to a voxel set that do not change if the topology of a set does not change when adding or removing voxels to the set. Therefore, we need to define the concept of topological equivalence and topology change. For these, we apply digital homotopy and cell complex theory.

A voxel set with vertex connectivity can naturally be considered as a cell complex, where the voxels and the voxel facets, edges, and vertices are *cells* of decreasing dimension. As such, any cell operation can be performed on the voxel set. In general, a *face* of a cell α is a cell β with dimension 1 less than the dimension of α. (Note that this is different from our usage of the term *face connected*; in the context of digital homotopy we use this more general definition.) A *free face* of a cell is a face that is not shared by any other cell in the complex. Two basic operations on a cell complex

are the *cell collapse* and *cell expansion*. A cell α may be collapsed through a face β if β is a free face. A new cell complex is formed by removing α and β, and the new complex has the same topology as the old complex, because the cell collapse can be seen as a deformation retraction. The inverse operation is the cell expansion. A cell α can be added to a complex if all of its faces of one dimension lower already exist in the complex, except one. This becomes a free face in the new complex. This is how voxel topology is related to simple homotopy theory [19]. For a face-connected voxel set we may apply cell collapses and expansions to the complement. The paper "Homotopy in Digital Spaces" [3] gives a thorough formal description of this in a general context and gives an example of a theorem one can prove with it.

Viewing the voxel set as a cell complex gives a very neat description of topological equivalence between sets. (Technically this is homotopy equivalence, but we use it for topological equivalence in this setting.)

Definition Two vertex-connected voxel sets A and B are *topologically equivalent* if, when viewed as cell complexes, there is a sequence of cell expansions and collapses that transforms A into B. Two face-connected voxel sets A and B are topologically equivalent if $\overline{A}$ and $\overline{B}$ are topologically equivalent.

Clearly the reverse sequence takes B to A if the expansions and collapses are switched. See Kaczynski et al. [58] for how to use this definition to compute topology of voxels sets in general.

In three dimensions, a trivial implementation of the voxel set as a cell complex requires a representation that is 8 times the size of the original voxel data, since every voxel, facet, edge, and vertex needs to be represented. Instead of using a more complex data structure (as in paper [8]), we use a slightly more complex operation, the voxel carve.

Definition The *voxel carve* is the discrete operation that can be performed on a

voxel set. It indicates removal of one voxel from the voxel set. The complementary operation is called a *voxel add*, and it is equivalent to a carve on the voxel set complement.

Definition A *local topology change* is a voxel carve (or add) that is not equivalent to a sequence of cell collapses and expansions. For a face-connected voxel set, a local topology change is a voxel carve (add) that is not equivalent to a sequence of cell expansions and collapses in the complementary cell complex.

The key to analyzing the topology of cubical scalar fields is this fact:

Theorem 3.1.1 *Let S be a voxel set, and let S' be derived by carving a voxel from S. Then S and S' are topologically equivalent if the carve operation is not a local topology change.*

The proof of this theorem follows from viewing the topology-preserving voxel carve as a strong deformation retraction [40] (see Figure 3) and as a sequence of cell collapses (see Figure 4). Note that the converse is not true; it is possible in certain situations for the global topology to stay the same when the local topology changes (see the description of the complex case below).

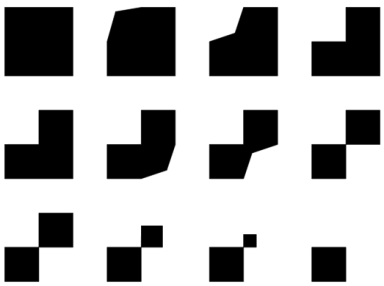


Figure 3: Three voxel carves, viewed as a deformation retraction. No topology changes.

Let S be a vertex-connected voxel set. A topology-preserving voxel carve in S is equivalent to a sequence of cell collapses, namely, collapsing the voxel and all its incident facets and edges that are not shared by other voxels in S. Whether a

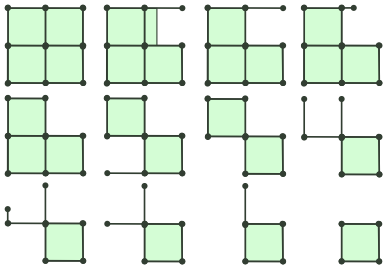


Figure 4: Three voxel carves, viewed as a sequence of cell collapses. No topology changes.

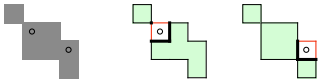
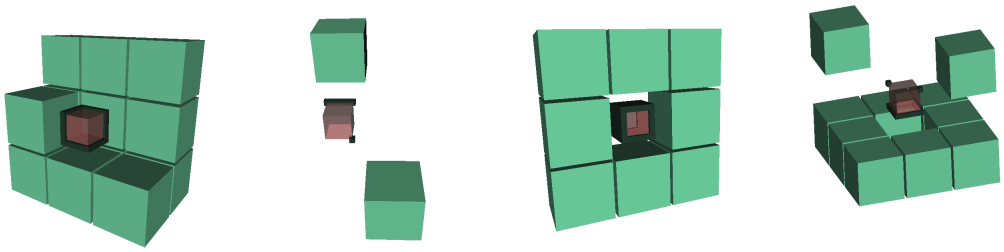


Figure 5: A 2D example of the local check with vertex connectivity. The leftmost element in the figure is the current set of voxels S, and the marked voxels are v_1 and v_2, two of the voxels we are considering for removal. The middle element shows that removing v_1 will cause a topology change since there are 2 components of the voxel boundary labeled inside (black) and 2 labeled outside (red). The rightmost element shows that removing v_2 will not cause a topology change since there is exactly 1 component of each label.

voxel $v \in S$ can be carved (without changing topology) is completely dependent on whether v's neighbors are in S or not, since they determine whether v and its faces have free faces. We examine v's boundary and its intersection I with the voxel set. Let $I = |S - \{v\}|_{26} \cap \partial v$. We count the components of I and $\partial v - I$, and for ease of notation, we classify v with a pair (i, o), where i is the number of components of I and o is the number of components of $\partial v - I$. See Figure 5 for a two-dimensional application of this test.

If v is classified $(1, 1)$, then v can be carved without changing the topology of the of the voxel set. See Figure 6a for this case in three dimensions.

Other cases make changes to the topology of S in the form of deleting or creating cycles. For example, if $i = 2$ (see Figure 6b), then either a 0-dimensional cycle is created (one component of S splits into two) or a 1-dimensional cycle is deleted (one handle of S is broken). Either way the Euler characteristic increases by 1. Note that



(a) (b) (c) (d)

Figure 6: Some three-dimensional cases of the local check with vertex connectivity. Each diagram shows the voxel v we are considering for removal and its $3 \times 3 \times 3$ neighborhood in S'. The boundary of v is labeled inside (black) and outside (red). a. No topology change $(1, 1)$. b. A thread is broken $(2, 1)$. c. A patch is punctured $(1, 2)$. d. A complex case—a thread is broken and a patch is punctured $(3, 2)$.

we cannot distinguish between these two possibilities using a local check, unless we have some prior knowledge of the global topology of S. For example, the number of 1-dimensional cycles cannot decrease past 0, so if S has no handles, $S - v$ must have one more component than S in this case. As another example, consider the case that $o = 2$ (see Figure 6c). Now either a 1-dimensional cycle is created (a new handle is punched through) or a 2-dimensional cycle is deleted (two components of $\overline{S}$ merge). In this case the Euler characteristic decreases by 1. Again which of the two occurs we cannot in general know using a local check. Note that these two cases ($i = 2$ and $o = 2$) may occur simultaneously (and the Euler characteristic would not change). We call this (and any case where $i + o > 3$) a complex case because more than one change is occurring. It may be that one handle is broken and another is created, resulting in no change in the Betti numbers of the voxel set, but since cycles are created and destroyed this is still considered a topology change. See Figure 6d for an example of a complex case.

The topology test for adding a voxel v to a vertex-connected set is the same, but of course $I = |S|_{26} \cap \partial v$. If S is a face-connected set, we use the same test but on the complement of S. To test carving a voxel from S, we use $I = |\overline{S}|_{26} \cap \partial v$, and to test adding a voxel to S, $I = |\overline{S} - \{v\}|_{26} \cap \partial v$.

The voxel boundary ∂v can be seen as a multipartite graph that describes the incidence relationship between faces of differing dimensions (see Figure 7). The graph is partitioned into two subgraphs based on I and $\partial v - I$, and the number of components of each of these subgraphs is counted. To keep the procedure efficient, the counting process can quit as soon as more than one component (of either set) is found. In theory the test is constant time, since the incidence graph is fixed for a given dimension. The size of the incidence graph (number of edges and vertices) of a d-dimensional hypervoxel is $3^d + 2d3^{d-1} - 2d - 1 = (3+2d)(3^{d-1}-1)+2 = O(d3^d)$. In particular, for 2, 3, and 4 dimensions this number is 16, 74, and 288. Since counting the connected components in the graph does take some time, for small dimensions we use look-up tables for the local topology check and its variants. The look-up tables for 2 and 3 dimensions, respectively, have 2^8 and 2^{26} entries. These are manageable numbers, but 2^{80} (for four dimensions) is not. Therefore, in four dimensions we have to count the components of the voxel boundary every time the test is performed.

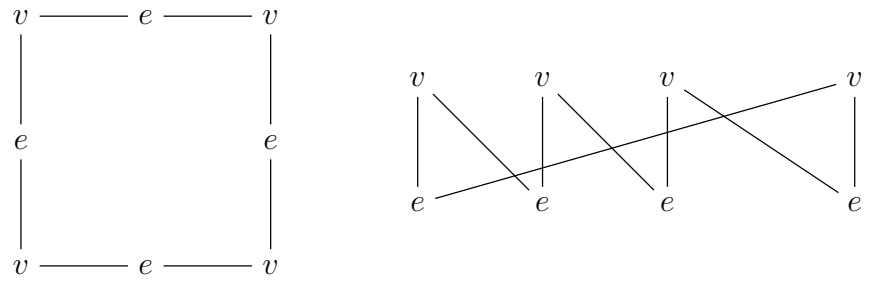


Figure 7: The incidence graph of the boundary of a 2D voxel. Left depicts the spatial relationship of the vertices (v) and edges (e), and right depicts the relationship as a multipartite graph.

In three (and lower) dimensions, having one component of I and one component of $\partial v - I$ is enough to guarantee no change in topology, since each component must be a topological disc. However, in four dimensions, the two components may be tangled in a topologically nontrivial way (for example, see Figure 8). Therefore we have to compute the Betti numbers of I as Delfinado and Edelsbrunner [24] do. We do this

by computing the Euler characteristic of I. If it is 1, and I and $\partial v - I$ each have one component, then I is topologically equivalent to a point. Gau and Kong [40] prove that carving v in this condition is equivalent to a strong deformation retraction from $|S|_{80}$ to $|S - \{v\}|_{80}$. Niethammer et al. [68] prove that this condition is sufficient for four (and lower) dimensions because computing the homology of I is equivalent to computing the Betti numbers of I. However, they were interested in computing the homology of I rather than the homotopy, so they do not guarantee the existence of a sequence of cell collapses.

```
* * *   * * *   * * *
* · *   * · *   * · *
* * *   * * *   * * *

* * *   * * *   * * *
* · ·   * o ·   * · ·
* * *   * * *   * * *

* * *   * * *   * * *
* * *   * * *   * * *
* * *   * * *   * * *
```

Figure 8: The $3 \times 3 \times 3 \times 3$ neighborhood of a 4D voxel v, represented as a 3×3 grid of 3×3 grids. Each cell is a face of v. Faces that belong to I are labeled with $*$ and faces that belong to $\partial v - I$ are labeled $\cdot$. The voxel (marked o) is changing from $*$ to $\cdot$. Note that face incidences are along axis-aligned directions and also move from grid to grid. I and $\partial v - I$ only have one component each, but they are not topologically trivial. $\partial v - I$ is a simple cycle. Changing o to $\cdot$ (i.e. carving v) results in a topology change because the cycle is filled in.

3.2 Topological Invariants

The first topological invariant we can examine in a voxel set S is the number of components.

Definition A *component* of a voxel set $\langle S, c \rangle$ is maximal, connected subset of $|S|_c$.

Using this definition, S may be partitioned into components, with each voxel of S in exactly one of these components. The number of components is a topological

invariant—it cannot change by voxel carving or adding unless the operation is a local topology change. This number is β_0, the 0th Betti number, the number of independent cycles of dimension 0 (i.e. points).

The next topological invariant is the number of cavities of a voxel set S.

Definition A *cavity* of a voxel set is a bounded component of the complement of the voxel set.

In d dimensions, the number of bounded components of the complement of S is β_{d-1}, so in three dimensions, this is β_2, the second Betti number. For a bounded voxel set B, the number of cavities of B is β_2.

In two dimensions this is the whole story—a topology change will change either the number of components or the number of cavities of the voxel set. In three dimensions there is another topological statistic, the number of handles. This is β_1, the first Betti number, the number of independent 1-cycles. Defining a handle precisely is difficult, but the handles in a voxel set seem intuitively clear. We can compute the number of handles using Euler's formula.

Definition The *Euler characteristic* χ of a bounded 3D voxel set is the number of vertices minus the number of edges plus the number of facets minus the number of voxels (parallelepipeds). $\chi = V - E + F - P$.

Note that this includes all vertices, edges, and facets of all the voxels in the set, and each is counted only once even if it is shared by multiple voxels. For a face-connected set, a vertex, edge, or facet is included only if all voxels it is incident to are in the set, and for a vertex-connected set it is included if any incident voxels are in the set. Note that a topology-preserving carve or add cannot change the Euler characteristic, since the operation is equivalent to a sequence of cell collapses. Clearly a cell collapse cannot change the Euler characteristic, since a cell collapse always involves an adjacent pair in this alternating sum.

We compute χ for an unbounded voxel set U by intersecting it with a large bounding box B that contains the complement of U. U and $U \cap B$ are homotopy equivalent because every voxel in U not in $U \cap B$ can be collapsed to the nearest face on the boundary of B. Since U and $U \cap B$ are homotopy equivalent, they have the same Euler characteristic.

Theorem 3.2.1 *For a 3-dimesional voxel set S, the Euler characteristic $\chi = \beta_0 - \beta_1 + \beta_2$.*

Using this result, we define the number of handles as follows:

Definition The *number of handles* of a voxel set is $\beta_0 + \beta_2 - \chi$.

Note that we are not defining what a handle actually is, although the meaning is intuitive: A torus has one handle, a double-torus has two handles, etc.

Thus, for a voxel set, we have three numbers that describe the topology of the set: the number of components, the number of cavities, and the number of handles. These are topological invariants; they cannot change after a voxel carve unless the carve is a local topology change.

Definition The *topological complexity* of a voxel set is the sum of the Betti numbers, i.e., the number of components plus the number of handles plus the number of cavities. A *topological feature* of a voxel set is a component, a handle, or a cavity.

A central idea in this thesis is the *importance* of a topological feature. We do not define this precisely because its meaning is subjective and depends on the user of the data. Intuitively, a topological feature is important if it is a part of the original object being modeled on the computer. If the feature is a result of the process that created the computer model, it is not important; it is topological noise.

3.3 Isosurfaces

One way to visualize a voxel set is to construct a mesh that represents its boundary. To do this we appeal to the language of scalar fields.

Definition A *scalar field* is a continuous function that assigns a scalar value to each point in R^3. Generally, a scalar field is a pair $\langle F, h \rangle$ where $h : F \rightarrow R$ is called the *height function*.

Definition An *isosurface* (or *level set*) of a scalar field $\langle F, h \rangle$ is the set $\{x \in F | h(x) = c\}$, where $c \in R$ is a given *isovalue*. A *contour* is a connected component of a level set. A *sublevel set* is the set $\{x \in F | h(x) \leq c\}$. A *superlevel set* is the set $\{x \in F | h(x) \geq c\}$.

A cubical scalar field can be seen as a a discrete sampling of a continuous scalar field. Alternatively, one can define a scalar field from a cubical scalar field using a rule for interpolation.

Definition A cubical scalar field and a given isovalue define two voxel sets, an *inside* set and an *outside* set. The inside set is the set of all voxels whose value in the cubical scalar field is less than the isovalue. We consistently (but arbitrarily) use vertex connectivity for inside sets in this thesis. The outside set is the complement of the inside set.

Note that an arbitrary vertex-connected voxel set V can be seen as the inside set of some cubical scalar field by assigning a value of -1 to every voxel in V and a value of 1 to every voxel in the complement of V, and using 0 as the isovalue.

We wish to extract an isosurface from a cubical scalar field, and so we require an isosurface extraction method that meets certain topological constraints. First of all, the isosurface must separate the centers of the inside voxels from the centers of outside voxels. When this holds, we may define the *inside* of an isosurface to be the

set of all points $p \in R^d$ such that there is a path in R^d that connects p to an inside voxel center without crossing the isosurface. Now let I_S be the union of the extracted isosurface and its inside, and let I_V be the realization of the inside voxel set. The method of isosurface extraction must ensure that I_S and I_V have the same homotopy type, both locally and globally.

Marching cubes [64] creates a triangle mesh from a 3D cubical scalar field and an isovalue on a cell-by-cell basis. In the context of isosurface extraction, the term *cell* refers to a smallest cube with voxel centers for vertices. Each cell has 2^d vertices. Marching cubes constructs vertices for the triangle mesh on cell edges that connect centers of adjacent voxels that are on opposite sides of the isovalue (one voxel is in the inside set and one is in the outside set). See Figure 9. The location of each vertex along the edge is determined by linearly interpolating the isovalue between the voxel values. Marching cubes then constructs triangles to connect the mesh vertices. Since the connectivity of the triangles does not depend on the location along each edge of the vertices, marching cubes uses rules for the construction of the triangles based only on which cell edges have mesh vertices. There are only a finite number of cases [4].

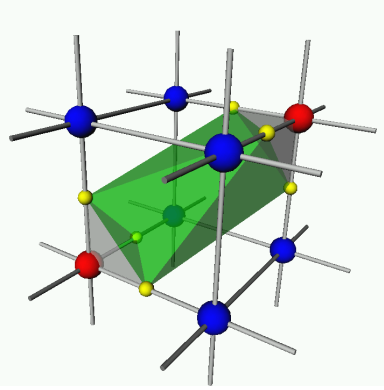


Figure 9: One possible case for marching cubes. The marching cubes algorithm constructs triangles (green) and vertices (yellow spheres). The spheres represent voxel centers. Red voxels are below the isovalue and blue voxels are above.

For marching cubes to work for our isosurface algorithm, the rules for connecting the mesh vertices with triangles must provide the desired local homotopy equivalence.

One way to ensure this is to require that both I_S and I_V have strong deformation retractions to $I_S \cap I_V$ in which points in a cell are mapped to the same cell. In particular, the topology of I_S in each cell will be the same as the topology of I_V in the cell. The topology of I_V is determined by the choice of voxel connectivity, which is vertex connectivity for an inside set. Therefore, the inside of the isosurface formed by marching cubes should also connect centers of voxels that share a vertex. See Figure 10 for the 2D equivalent, marching squares.

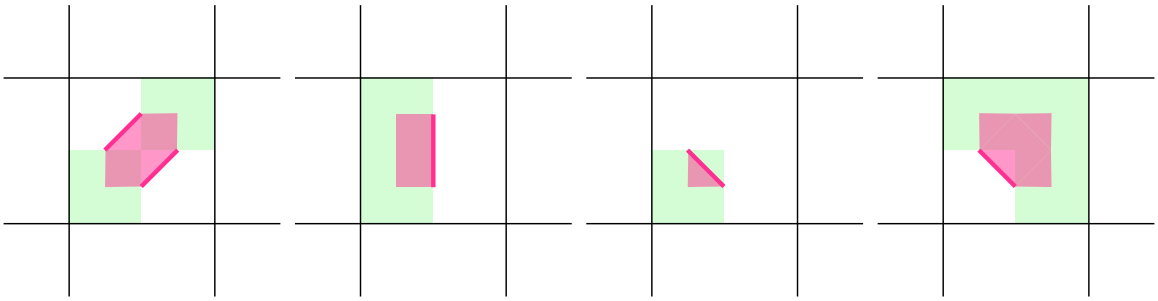


Figure 10: Cases for marching squares. Each element is one cell (only one quarter of each voxel is visible). Green is I_V in the cell and pink is I_S in the cell. Note that both sets have a deformation retraction to their intersection.

Bhaniramka's variant of marching cubes [6] creates the rules for constructing triangles assuming that the mesh vertices are centered on cell edges. In each cell, they construct the convex hull of all vertices of the mesh and centers of inside voxels. The convex hull is triangulated and only the triangles on the interior of the cell are used. These rules connect the centers of vertex-connected inside voxels, and therefore this method meets all of our needs for an isosurface extraction algorithm.

If the inside set of a 3D cubical scalar field is bounded, the isosurface is a bounded 2D manifold with no boundary (provided the isovalue does not take on the value of any voxel). The number of components and the number of handles of these components are derived exactly from the topology of the inside and outside sets.

As the isovalue changes, the isosurface of the cubical scalar field changes as well. We assume that no voxels have the same value so that all changes are isolated. When this is not the case we may perturb the voxel values slightly so they are all distinct.

Now, let s_1 be the value of some voxel v, and let s_0 and s_2 be given such that $s_0 < s_1 < s_2$ and no other voxels have value between s_0 and s_2. Consider the inside set S defined by s_2 and the inside set S' defined by s_0. Then changing the isovalue from s_2 to s_0 is equivalent to carving v. In fact, the isosurface changes local topology if and only if v fails the local topology test in S (that is, the number of components of I or $\partial v - I$ is not 1).

Definition As the isovalue is varied, a voxel v may change from inside to outside (or vice versa). If this is a topology change in the inside set (if the local topology test for v fails), the voxel is called a *critical voxel*.

Note that this is analogous to a critical point in a scalar field $\langle F, h \rangle$, where h is a smooth Morse function. In this context a critical point is a point where the isosurfaces change local topology. That is, $x \in F$ is a critical point if for an ϵ-ball around x, isosurfaces just above and just below $h(x)$ have different topology. This follows from the classical Morse theory definition of a critical point, a point with 0 gradient.

Definition The *topological complexity* of a cubical scalar field is the number of critical voxels. The term *topological feature* when applied to a cubical scalar field is synonymous with critical voxel.

The reuse of this term (topological complexity of a cubical scalar field) is reasonable because the critical voxels are where the Betti numbers (topological complexity of a voxel set) of the inside voxel set change as the isovalue changes.

Again we provide an intuitive description of the *importance* of a topological feature. Consider a cubical scalar field and a steadily increasing isovalue. If a component of the inside set appears and then quickly merges with another component over a small interval of isovalues, then the critical points associated with the appearance and merging are probably not important. On the other hand, if a handle is formed in the inside set (i.e. the number of handles increases), and only after a large interval of

isovalues the handle is filled in, then the critical points associated with the creation and filling in of the handle are important.

3.4 Contour trees

The contour tree is a topological representation of a cubical scalar field—it describes how components of the isosurface (contours) join and merge as the isovalue changes. As such, it does not contain any geometric information. This is a limitation in the sense that data is lost, but the information that it does keep is useful for a wealth of possibilities. Furthermore, the small representation makes storage and processing more efficient. Another advantage is that subdomains of the volume data may be independently analyzed followed by simple merging of the contour trees for observation of global phenomena.

We first define contour trees in the context of scalar fields and then analogously in the context of cubical scalar fields. Finally, we describe scalar fields and contour trees in terms of category theory and use point-set topology to make the descriptions precise.

Definition The *contour tree* of a scalar field is the quotient space defined by identifying to a point each component (or contour) of each level set in the domain.

If the domain of the scalar field is simply connected, then the quotient space has no cycles, and for this reason we call it a tree. Otherwise it is called a *Reeb graph* [80]. The combinatorial (graph) structure is applied by defining equivalence classes of contours that are nearby in isovalue and contain no critical points of the scalar field. The critical points themselves are the vertices in the graph, and the equivalence classes form the edges. Definitions and applications are explained thoroughly in Carr's Ph.D. thesis [13]. See Figure 11 for a 2D example.

We could compute the contour tree of a cubical scalar field C by first extending C to an appropriate continuous scalar field and then taking the quotient space. However,

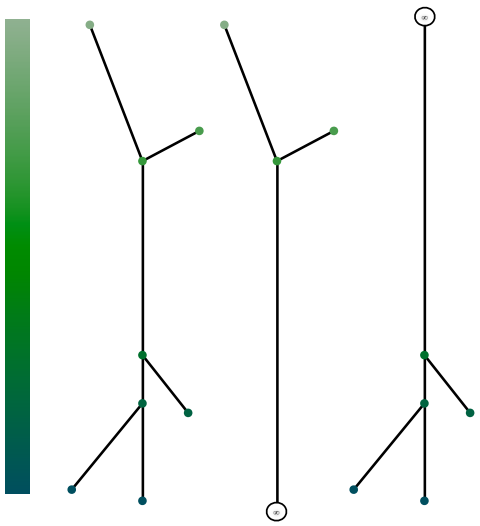


Figure 11: The contour tree, split tree, and join tree (left to right) for the 2D cubical scalar field show in Figure 2.

we avoid converting to the continuous case and instead define the contour tree of C algorithmically. The algorithm we use for computing the contour tree, described by Carr et al. [14], involves the construction of two intermediary trees, the split tree and join tree. These constructions process the data in order according to sample value. The join tree is computed by keeping track of all the components of the voxel set below the current isovalue (the inside set). As the isovalue increases, new components may appear or components may join. As components appear, new leaf nodes in the tree are created. As components join at critical points, edges from their respective nodes are joined at a new internal node. The split tree is computed with a complementary algorithm that proceeds in the opposite order and creates nodes when components of the outside set appear and join. The "joining" and "splitting" refer to the contours, rather than what the trees actually depict, which is the components of the inside set and outside set, respectively. Finally a merging algorithm constructs the complete contour tree from these two trees.

We can define the join tree and split tree of a cubical scalar field C using a partial order of the voxels, assuming no two voxels have the same value. Define $C_{-}(v)$ to be the component of the inside set of C with isovalue just above the value of v (below

any voxel above v). For two voxels a and b, $a \leq b$ if $C_-(a) \subseteq C_-(b)$. The *join tree* of C is the result of a topological sort of the critical voxels of C with respect to the partial order $\leq$. Note that the result is a tree when C is simply connected. Likewise define $C_+(v)$ to be the component of the outside set of C with isovalue just below the value of v, and for two voxels a and b, $a \geq b$ if $C_+(a) \subseteq C_+(b)$. The *split tree* of C is the result of a topological sort of the critical voxels of C with respect to the partial order $\geq$.

Definition The *contour tree* of a cubical scalar field C is the result of merging the split and join trees. It is a pair of sets, the vertices and edges. The vertices are critical voxels of C that connect contours as the isovalue changes. The *lower edges* of a vertex v correspond to contours that join at v and the *upper edges* correspond to contours that split at v, as the isovalue increases.

An *augmented contour tree* is similar to a contour tree except that it may have vertices that have one upper neighbor and one lower neighbor. Note that adding such vertices does not change the topological structure of the tree. One use for such nodes is the method described by Pascucci et al. [73] to compute the Betti numbers of the components of the sublevel sets. Since the Betti numbers can change not only when components appear and merge but also when a single component contacts itself (a handle is formed), nodes are added to the tree in all of these cases. The Betti numbers can then be computed correctly using the augmented contour tree. Berglund and Szymczak [5] construct a "subdomain-aware" contour tree from a domain D and a subdomain S by augmenting the contour tree of D with all critical points in D and in S in order to define a map from the contour tree of S into the contour tree of D.

Pascucci et al. [73] describe a divide-and-conquer approach to computing the split and join trees. Any domain may be divided into two subdomains. The join and split trees are computed for these subdomains and then merged according to the shared

boundary of the subdomains to build the join and split trees of the full domain. The base case is the set of 8 voxels that share a single vertex. The contour tree for the base case depends only on the relative values of the 8 voxels, so there are a finite number of cases. Pascucci et al. [73] describe all the possible cases.

3.4.1 Calculating topological invariants

We next describe how to calculate the topological invariants for the inside sets that are represented in the join tree of a 3D cubical scalar field. The outside sets represented in the split tree are complementary, so the same approach can be used. Each edge in the join tree (augmented by all critical points) corresponds to an inside set that does not change topology as the isovalue changes. Therefore the topological invariants do not change along the edge, and so we label each edge with the Betti numbers of the corresponding inside set. Pascucci et al. [73] compute the Betti numbers of a simplicial mesh using the contour tree. We need the Betti numbers of a voxel set, so we label the edges of the join tree (and split tree) instead.

Each edge corresponds to a single component, so $\beta_0 = 1$. To compute β_2, we use an inclusion-induced map inspired by Berglund and Szymczak [5]. For a given isovalue, each contour C is mapped into the sublevel set component C_- that contains it. Note that many contours will map to C_- when C_- has cavities. This map induces a map from the contour tree onto the join tree. The number of edges of the contour tree that map to an edge e of the join tree is $\beta_2 + 1$, so we have β_0 and β_2 for e.

To compute β_1, we use the Euler characteristic. Every edge incident to a leaf in the join tree has $\chi = 1$, since it is topologically equivalent to a single voxel, a local minimum. For each of the other edges, consider the vertex at the low end of the edge. Since it is a critical point, components may be merging or a component may be contacting itself. No matter which case, the new voxel set (as the isovalue increases from below the critical voxel to above it) is the union of the voxel sets represented by

edges coming into the vertex from below and the critical voxel v. Therefore the Euler characteristic is simply the sum of the Euler characteristics of the components that are joining (already calculated) plus the Euler characteristic of v $(8-12+6-1=1)$ minus the Euler characteristic of the intersection of v with the union of the joining sets (since it is counted twice in the union). This intersection is the set I described above in Section 3.1. Calculating the Euler characteristic of I is easier than the test for topology change because we only need to count the vertices, edges, and facets in I; we do not need to calculate connected components. Thus χ can be computed for each edge in the join tree starting from the leaves. As stated above, $\beta_1 = \beta_0 + \beta_2 - \chi$.

3.4.2 Category of scalar fields

In order to create a theoretical underpinning for scalar fields, contour trees, and their associated maps, we propose to define $\mathcal{S}$, the category of scalar fields. A scalar field is a continuous real-valued function $h : X \to R$ defined on a topological space X. h is called the height function. The morphisms of $\mathcal{S}$ are the height-preserving maps. Let h_X and $h_{X'}$ be objects of $\mathcal{S}$, height functions defined on spaces X and X', respectively. Then a morphism i_f of $\mathcal{S}$ takes h_X to $h_{X'}$ if i_f defines a continuous map $f : X \to X'$ such that for all $x \in X$, $h_X(x) = h_{X'}(f(x))$. Composition of morphisms is defined by composition of the underlying continuous maps on the underlying topological spaces. Associativity follows from associativity of function composition. The identity morphism 1_{h_X} defines the identity map $f(x) = x \ \forall x \in X$.

Note that, in particular, inclusion maps are morphisms of $\mathcal{S}$. If h_X and h_Y are objects of $\mathcal{S}$ such that $Y \subseteq X$ and $\forall y \in Y \ h_Y(y) = h_X(y)$, the map $f : Y \to X$ given by $f(y) = y$ is defined by a morphism of $\mathcal{S}$, since inclusion maps on topological spaces are continuous and the height is preserved.

Also note that any graph may be assigned a topology by embedding it in R^3. Thus a continuous real-valued function defined on the embedded graph is an object

of $\mathcal{S}$.

3.4.3 The contour tree functor

We now define a functor that takes $\mathcal{S}$ to $\mathcal{S}$ and maps an object $h : X \to R$ to its contour tree. A contour of h is a connected component of the level set $h^{-1}(w)$, the set of all points in X whose height value is w, $w \in R$. Recall that a connected component of a set $A \subseteq X$ is a subset of A that is both open and closed with respect to A and has no proper subset (other than $\emptyset$) that is both open and closed. Let X^* be the partition of X into all the contours of h, i.e. one point in X^* for each component of $h^{-1}(w)$ for each value of w. (If $h^{-1}(w) = \emptyset$ there are no components for that isovalue.) Let $p : X \to X^*$ be the surjective map that takes $x \in X$ to the contour $C \subseteq X$ where $x \in C$. The contour tree is the quotient space defined by p, that is, the set X^* with topology defined by the quotient map p. Note that each point in the contour tree, a point in X^* and a contour C in X, has an associated height value $w = h(C)$. Therefore the contour tree is a topological space with a height function defined on it; call this $h' : X^* \to R$. Note that $h'(p(x)) = h(x)\ \forall x \in X$. Let A be an open set in R. Then $h^{-1}(A)$ is open in X, since h is continuous. Also, $h^{-1}(A) = p^{-1}(h'^{-1}(A))$. Since p is a quotient map, $h'^{-1}(A)$ must also be open. Thus h' is continuous and therefore an object of $\mathcal{S}$.

Let h_X and h_Y be objects of $\mathcal{S}$, and let $f : X \to Y$ be the height-preserving map defined by a morphism i_f from h_X to h_Y. Let X^* and Y^* be the contour trees of X and Y, respectively, with associated quotient maps p_X and p_Y and associated height functions $h_{X^*} : X^* \to R$ and $h_{Y^*} : Y^* \to R$. Let $C_X \in X^*$. Then $p_X^{-1}(C_X)$ is a contour in X, i.e. a connected component of the level set $h^{-1}(h_{X^*}(C_X))$. Since f is continuous, $f(p_X^{-1}(C_X))$ is also connected, and since f is height-preserving, $h_Y(f(p_X^{-1}(C_X))) = h_X(p_X^{-1}(C_X)) = h_{X^*}(C_X)$. Thus $f(p_X^{-1}(C_X))$ is a connected set contained in a level

set in Y, and therefore $p_Y(f(p_X^{-1}(C_X)))$ is a single point in Y^*. So we define a height-preserving map $f^* : X^* \to Y^*$ as follows: $f^*(C_X) = p_Y(f(p_X^{-1}(C_X)))$. Thus i_{f^*} is the result of applying the contour tree functor to the morphism i_f, and thus the contour tree functor takes height fields to height fields and height-preserving maps to height-preserving maps and is therefore well defined. See Figure 12.

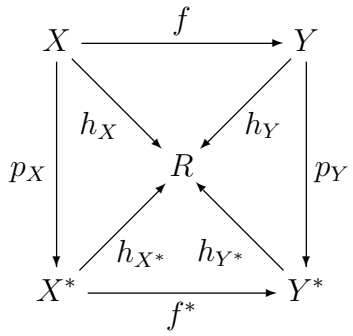


Figure 12: This commutative diagram depicts how we are able to define the map between contour trees induced from a height-preserving map f between the scalar fields X and Y.

We call this the contour tree functor, but it really computes the Reeb graph [80]. The fact that X^* is one-dimensional (and can therefore be considered a graph) follows from classical Morse theory. Also, if X is simply connected, then the graph X^* is actually a tree. This follows from Carr's theorems [13] about the association of paths in X and paths in the contour tree. Finally, note that the continuous versions of the join tree and split tree are also functors on $\mathcal{S}$.

3.4.4 Subdomain-aware contour trees

We can use the category and functor to describe the subdomain-aware contour trees described by Berglund and Szymczak [5]. Let A be a subdomain of a domain X, and $f : A \to X$ be the inclusion map from A to X (that is, $f(x) = x$ for all $x \in A$). Then f^*, defined by the contour tree functor, is called the *inclusion-induced map*. It is a map from the contour tree of A to the contour tree of X.

Consider a three-dimensional cubical scalar field defined as the distance to a V (see Figure 13). Let X be this domain, and let A be one slice of domain (a 2D cubical scalar field). X has only a global min and a global max, so the contour tree is just an edge and two nodes. However, A has two minima and a global max, so its contour tree has a branch. To construct the subdomain-aware contour tree of X, we first augment the contour tree of X with the critical points of A. Then we use the inclusion-induced map to count how many edges of the contour tree of A are mapped to the augmented tree of X (see Figure 14).

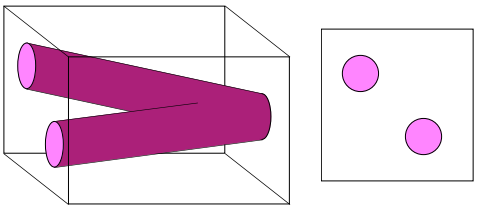


Figure 13: Isosurfaces from the domain X (left) and the subdomain A (right), a 2D slice of X.

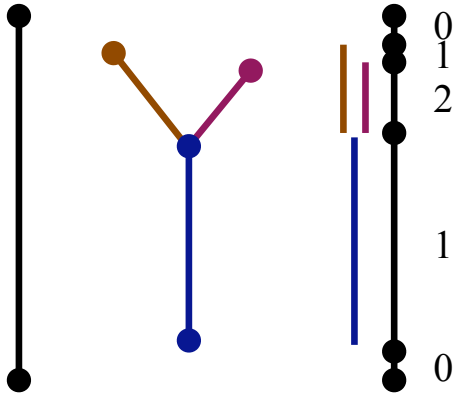


Figure 14: From left to right, the contour tree of X, the contour tree of A, and the subdomain-aware contour tree with labels from the inclusion-induced map.

The inclusion-induced map and the subdomain-aware contour tree give useful information about the relationship of the contours in the domain and the subdomain. In this example, one can easily see from the subdomain-aware contour tree that the contour has two components on the boundary. Further slicing would reveal that the

contours in the slices merge sooner (at smaller isovalues) as you move through the domain.

3.4.5 Recursive decomposition

Pascucci et al. [71] developed an algorithm for recursive construction of contour trees on simplicial complexes and rectilinear grids. The recursive decomposition can be stated in terms of the category and functor described above. Let h_A and h_B be members of $\mathcal{S}$ such that for all $x \in A \cap B$, $h_A(x) = h_B(x)$, and let A and B be closed in $X = A \cup B$. Then h_X, defined by $h_X(x) = h_A(x)$ for $x \in A$ and $h_X(x) = h_B(x)$ for $x \in B$, is an object of $\mathcal{S}$. If A^* and B^* are the contour trees of A and B, respectively, we may construct the contour tree X^* of X as follows. Consider the inclusion map i_A from $A \cap B$ into A. Note that i_A is a height-preserving map and therefore induces a height-preserving map i_A^* from the contour tree of $A \cap B$ into A^*. Likewise let i_B and i_B^* be the inclusion map and induced map for B. Also, let j_A, j_B, and j_{AB} be the inclusion maps from A, B, and $A \cap B$ into X and let j_A^*, j_B^*, and j_{AB}^* be the associated induced maps. Note that each contour of A is contained in a contour of X and each contour of B is contained in a contour of X. Let x be a point in $A \cap B$ and C be the contour of $A \cap B$ containing x. Note that C is contained in a contour C_A of A and a contour C_B of B, as well as a contour of X. But this contour of X containing C must be same as the contour of X containing C_A and C_B, since all four of these contours share the point x. In other words, $p_X(x) = j_A^*(p_A(x))$ if $x \in A$, $p_X(x) = j_B^*(p_B(x))$ if $x \in B$, and $p_X(x) = j_A^*(p_A(x)) = j_B^*(p_B(x)) = j_A^*(p_A(i_A(x))) = j_B^*(p_B(i_B(x))) = j_{AB}^*(p_{A\cap B}(x))$ if $x \in A \cap B$, where the p functions are the quotient maps defined above. Thus we can form the contour tree of X by taking the contour trees of A and B and identifying the points mapped by i_A^* and i_B^* from each point in the contour tree of $A \cap B$.

CHAPTER 4

TOPOLOGY SIMPLIFICATION

Topologically simple models are important in many applications, such as texture mapping [76], simplification [39], compression, mapping between surfaces [67], and surface parameterization [96]. For example, there are many methods of mesh simplification, but if the user wants the simplification to preserve the manifold structure of the mesh, the topology of the mesh must also be preserved. An example of this is Hoppe et al.'s [51] mesh simplification using edge contraction. The topology preservation can be desirable except when there is topological noise that the user wishes to be simplified away. Fortunately, our technique removes all the small handles but retains the important topological features. Thus, when topology simplification is used as a preprocess to mesh extraction, there is no need to worry about getting bogged down in the topological noise during mesh simplification.

4.1 Isosurfaces

A cubical scalar field and an isovalue define a voxel set below the isovalue, the inside set. This set has a certain topology. In order to simplify the topology, we wish to edit the cubical scalar field so that the inside set has the desired topology, but only regions associated with the simplifications are changed. The idea is to start with a voxel set S of known topology and transform it into the inside set. The transformation is done with voxel operations while paying attention to the local topology changes.

For example, consider a cubical scalar field where the value of each voxel is the distance to a figure eight (see Figure 15). A certain isovalue h will result in an extracted isosurface like the one in the figure. The genus is 2. We start with a bounding box that contains all voxels below with value below h. The procedure then

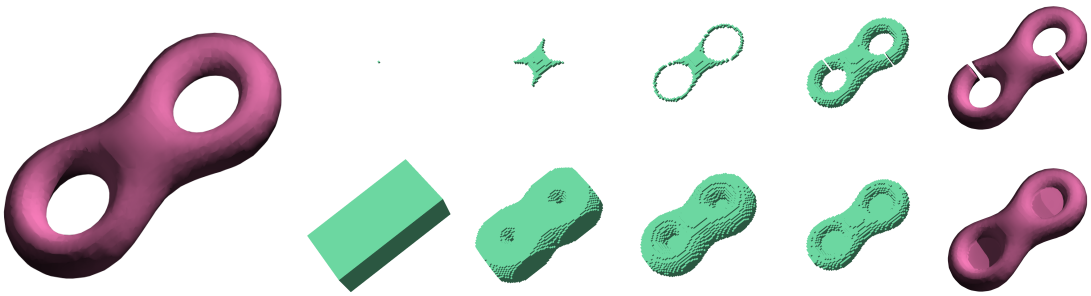


Figure 15: Progress of the carving procedure on a figure eight. Left is an extracted isosurface from the original volume. Top is adding from inside and bottom is carving from outside. Far right is the extracted isosurface in each case.

carves as many voxels with value above h as possible, without changing topology. When no more voxels can be carved, the values of the voxels above h that remain uncarved are changed to be slightly below h. The result is that the extracted isosurface now has patches over the handles. On the other hand, we can start from a single voxel contained in the inside set we wish to simplify. To this voxel we add voxels without changing topology and stop when no more voxels with value below h can be added. Any inside voxels that were never added are set to a value slightly above h, and the result is an isosurface in which both handles are broken. Both methods result in an isosurface that has trivial topology, that is, topologically equivalent to a sphere.

Both methods maintain a vertex-connected set S that is often referred to as "the inside set" in this thesis. For the method that carves voxels starting from a bounding box, S is the set of uncarved voxels. Since this method starts outside (above) the given isovalue and carves voxels from S, it is called "carving from outside." For the method that adds voxels to one initial inside voxel, S is the set of voxels already added, and the method is called "adding from inside." Note that "carving from outside" and "carving a voxel from an outside set" (Sections 3.3 and 3.1) have completely different meanings but are similar-sounding.

This is the basic method of volumetric repair of isosurface topology. The rest of this section elaborates on this technique. We discuss construction of models with

Figure 16: Results for the Happy Buddha model. Each image shows one more handle opened. The fourth handle opened is in the back, and it can't be seen from this perspective until the fifth handle is opened.

nontrivial desired topology, variants of the local test for different results, a multiresolution approach using octrees, and the effect of carving order.

4.1.1 Non-trivial topology

Often the isosurface we wish to simplify has important topological features that are not noise but are inherent to the modeled object. For example, the happy Buddha model (Figure 16) has genus 5. To construct a genus-5 surface, we start by applying the basic method to a signed-distance volume where the model is represented by isosurface 0. After the initial carving procedure has run to completion, we carve one topology-altering voxel. This results in either punching through a patch over a handle or breaking a thread between components. The carving of the topology-changing voxel is followed by the standard carving procedure that preserves the topology. This is repeated until a number (specified by the user) of topology altering operations has been performed. Thus, in typical cases the topological complexity of the resulting inside voxel set is the same as the number of topology-altering voxels that were removed

Atypical cases include complex topology changes. When carving (or adding) a voxel results in a complex topology change, it is possible for the topological complexity to increase by more than one or even decrease. The latter occurs when, for example,

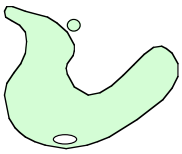


Figure 17: An isosurface with extraneous components.

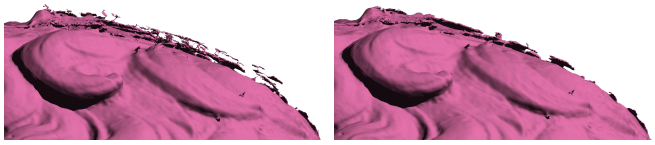


Figure 18: Motivation for the genus increase test, the top of David's head. Using Test A (left), all components of the original model are connected by threads. Test B (right) allows all small components to be thrown away.

a handle and a thread are broken at the same time. Other bad cases occur when topological obstacles (described in the Conclusion) are present in the data.

4.1.2 Test for genus increase

The local test used up to this point is a complete topology test—any voxel whose removal (or addition) changes the genus or the number of components of S or its complement cannot be carved. However, the result of applying this test (Test A) to isosurfaces with multiple components is not ideal. For example, scanned models such as the David head often have small unimportant components floating nearby the main component. See Figure 17 for a 2D example. This is also a form of topological noise—and carving using Test A indeed repairs the noise by forming a topological sphere—but it does so by creating threads between the components. For very small floating components this has an effect of creating hair-like appendages on the resulting surface (see Figure 18). The voxel-adding procedure has a complementary problem—any small bubble-like cavities inside the object are forced to connect to the outside through tunnels.

To prevent this problem, we handle separation of components differently from

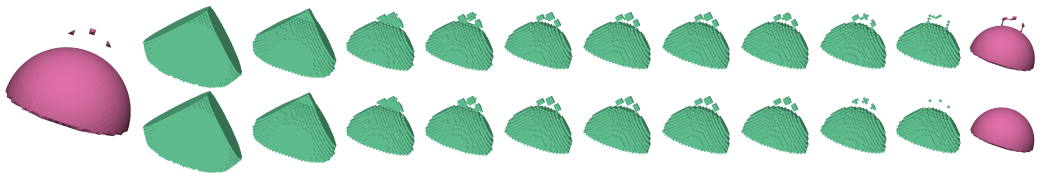


Figure 19: Progress of the carving procedure using the two topology tests. Left is an isosurface extracted from the input volume; note the floating components. The top row is the carving process using Test A and the bottom row is the carving process using Test B. Far right is the result of extracting the final single-component mesh from the outside set in each case.

changes in genus by testing for genus increases only (Test B). Since we start the carving process with a topological sphere (genus 0), no decreases in genus are possible. Recall from Chapter 3 that we partition the boundary of a voxel v into I, the intersection with the inside set, and $\partial v - I$. For ease of notation, we classify v with a pair (i, o), where i is the number of components of I and o is the number of components of $\partial v - I$. When v is classified $(2, 1)$ (Figure 6b), though Test A says this is a topology change, we know this is an increase in the number of components of the voxel set and not a decrease in genus. Therefore Test B allows v to be carved. Thus Test B allows splitting of components when carving from outside and, complementarily, cutting off of bubbles when adding from the inside.

When the surface mesh is extracted from the result of this procedure, it may have several components. Therefore, after extracting the isosurface we throw away all components except for the known components of the model. See Figure 19 for a comparison of results of Test B with the original.

Thus, the main differences between Test A and Test B are the following: Test A is the same for carving from outside and growing from inside—if v is classified $(1, 1)$, then carving or addition of v is allowed (Figure 6a, for example). Test B for carving from outside says that if v is classified $(x, 1)$ for any x then carving of v is allowed (Figures 6a and 6b). Test B for growing from inside says that if v is classified $(1, x)$ for any x then addition of v is allowed (Figures 6a and 6c). We thus need two look-up

tables for Test B—one for carving and one for adding.

4.1.3 Octree-based implementation

When processing large volumetric data sets such as the laser range data from Stanford [62], the need for compression is obvious. Since our algorithm is fairly spatially coherent, a spatial data structure such as the octree is the obvious choice. The principle is that the only information we really need is near the object surface. Therefore, any voxels sufficiently far from the surface may be merged into supervoxels. The basic tradeoff between space and time is as follows: Let n be the number of voxels in the largest dimension of the bounding box, so that there are a total of $N = O(n^3)$ voxels. Storing the whole volume takes $O(n^3)$ space, while storing the octree only takes $O(n^2)$ space, for reasonable models, since the surface should pass through on the order of n^2 voxels. On the other hand, voxel access takes constant time when the volume is stored directly and $O(\log n)$ time in the octree. However, in a careful implementation of the octree, certain voxel operations such as neighbor finding will be amortized constant time, since most voxel neighbors are nearby in the octree. See the octree book [75] for details. We chose not to implement this for simplicity's sake, but it could improve the time results somewhat because much of the time spent in the program (up to about one third of the total time) is on neighbor lookup.

We follow the standard bottom-up approach to constructing the octree—when all of a supervoxel's children have been seen, if they are all on the same side of the object surface they are merged into the supervoxel. However, we want to preserve any subvoxel information about the surface that is stored in the voxels. Therefore, we do not allow any voxel to merge that has a neighbor on the opposite side of the surface. Likewise, a supervoxel can only be merged with other supervoxels on the same level if it has no children, that is, if all of its children have been merged.

If marching cubes is used to extract the surface mesh, the subvoxel information is

interpreted as distance to the surface. Therefore, we use the subvoxel information to define analogous distance values for the supervoxels in the octree. This value depends on the state of the supervoxel's children. If all the children are inside the isosurface, then the parent is also considered inside, and the distance value is the minimum of the (absolute) distance values of the eight children. Likewise if all the children are outside then the parent is considered outside and the distance is the minimum of the child distances. Otherwise, the parent is mixed and is given a distance value of 0. This indicates that the isosurface passes through the supervoxel but the exact location is determined by the children.

An octree implementation is always a time-space tradeoff, but in our case the octree gives a very natural multiresolution implementation of the topology repair procedure. As such it is a tremendous speedup. The multiresolution implementation begins at the root level $h = \lceil \log_2 n \rceil$ (lowest resolution) in the octree and carves from there. Any supervoxel that has no children and does not cause a topology change may be carved. A supervoxel that does have children is too close to the isosurface to carve safely at this point, since the descendant voxels may contain subvoxel information. When all the supervoxels on the current level that can be carved have been carved, all the uncarved supervoxels are subdivided into their eight children for further carving at the next resolution step. This proceeds until level 0, the highest and finest resolution, and the deepest level of the octree. The result is that in practice only $O(n^2)$ topology checks and carving operations need to be performed instead of $O(n^3)$.

The payment for these time and space speedups comes in the form of quality of the result. For example, the patches over the handles may not be nice and flat in the multiresolution carving. If there is a large open handle in the model, and an octree cube happens to poke right through it, a patch will certainly be formed, but its shape will necessarily be defined by the large cube that was carved. See Figure 20. However, because the topological noise that we want to remove is typically very

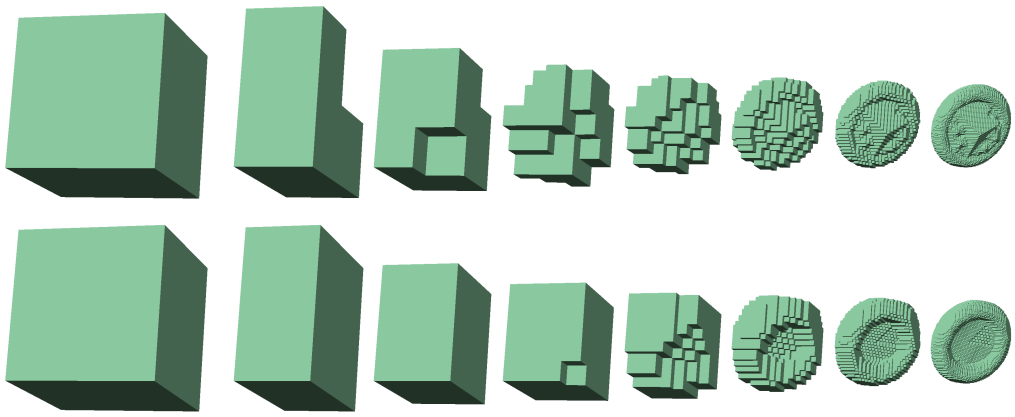


Figure 20: Carving with an octree. The top part of the figure shows the result of carving at the end of each octree level. The bottom part shows the same except that a layer of voxels is left behind at each resolution level. The benefit in smoothness is easy to see.

small, this is not too much of an issue. Furthermore, the tradeoff can be adjusted by leaving behind an extra layer of supervoxels at each resolution step. We implemented this extra layer as follows: After all the supervoxels on the current level that can be carved have been carved, we loop over all the carved supervoxels on this level. Any that have an uncarved neighbor (or a neighbor on the other side of the object surface) are candidates for uncarving since they are the supervoxels that form the shapes of the patches. These candidates are then uncarved if they do not cause a topology change. After the layer is uncarved, all uncarved voxels are subdivided as usual. This process allows finer-grained carving around topology adjustments and yields smoother results, but it comes at the cost of increased memory usage and increased running time due to the greater number of carving operations.

4.1.4 Carving order

The shape of the patches and cuts is controlled by the order in which the voxels are carved. (Here we use the term "carve" to refer to both operations—carving from the outside set and adding to the inside set—when there is no difference in the discussion.) We have already discussed how the low-resolution octree nodes can cut

through handles to make the patches rough. Other orderings have similar drawbacks. For example, the easiest voxel order to implement would be a lexicographic order. However, as carving proceeds, the patches that should be roughly flat get stretched in the direction of the carving. The topology of the result will still be correct, but the patches will look like cylinders instead of what we would prefer—nice discs across the handles. What we need is an ordering that meets the patch from both sides simultaneously, so we use the distance from the object surface as our order, as Aktouf et al. [1, 2] do. We have found that Manhattan distance is easiest to implement, but Euclidean distance gives smoother results.

For voxels adjacent to the isosurface (voxels with a neighbor on the opposite side), we interpret the original sample data as distance values. These values are used to seed the distance calculation. If no subvoxel information is present (i.e., the volume consists of binary data), we use a value of 0.5. The distance calculation then propagates outward and inward from the surface.

Once the distance values for all the voxels are computed, to determine the carving order we need to sort the voxels. However, a global sort is not necessary. During the carving, we maintain a priority queue that contains the voxels on the boundary of the current set of uncarved voxels (the "boundary queue"). The octree implementation requires only $O(n^2)$ carve operations on reasonable models, and each operation requires one extraction and a constant number of insertions on the priority queue, so the total running time in practice is $O(n^2 \log n)$, where n is the size of the largest dimension of the bounding box. When the multiresolution approach is not used, the running time is $O(N \log N)$, where N is the total number of voxels.

Calculating the distance function and prioritizing the voxels takes extra time, but aside from the nicer patches it also has the further advantage of providing an idea of the size of each handle. After all voxels that do not change topology have been carved, we arrange the topology-altering voxels in order of decreasing distance. If

the user desires the result to have genus greater than 0, we remove the most distant topology-altering voxel first and proceed with carving. This opens the handle that in some sense is the largest and therefore is most likely one of the important topological features the user wishes to preserve. This also allows the user to specify either how many handles to open or a threshold on the size of the handles. This measure of the size of each handle is similar to the idea of topological persistence in Edelsbrunner et al.'s fundamental work [27], so we believe it is a good measure of the appropriateness of including the handle in the output.

4.1.5 Algorithm Summary

Algorithm 1 FixSurfaceTopology

Require: A volume data set, an isovalue, and the desired number of features
Ensure: A watertight triangle mesh for the isosurface with simple topology

```
1: Construct an octree using the given isovalue
2: for each octree level from the root to the highest resolution do
3:    Calculate distances
4:    Initialize boundary queues for inside and outside sets
5:    CarveAndAdd
6:    if octree level is lower than the highest then
7:       Subdivide uncarved and unadded voxels on current level
8: for f = 1 to the number of features do
9:    Open feature on the inside set
10:   Open feature on the outside set
11:   CarveAndAdd
12: if user desires inside set then
13:   Fix volume inside
14: else
15:   Fix volume outside
16: Smooth in the voxel domain if desired
17: Extract isosurface with Marching Cubes
18: Output component with the most triangles
```

Algorithm 1 shows a formal description of our octree-based implementation. The input is a volumetric data set, an isovalue of interest, and the desired number of topological features. The output is a triangle mesh that represents the desired isosurface

Algorithm 2 CarveAndAdd

```
1: while there are uncarved voxels on the boundaries do
2:    Check both boundary queues to find voxel v with max distance
3:    Remove v from queue q
4:    if adding/carving v does not cause a genus change and v does not have children
      in octree then
5:       Add or carve v, depending on which boundary q is
6:       for all n such that n is a 26-neighbor of v do
7:          if n is not already added/carved and not already on the boundary and n
            is on the same side as v or undefined then
8:             add n to the boundary queue q
9:    else
10:      if v is on the highest octree level then
11:         Save v for opening features later
```

in the volume. This mesh is watertight and has genus equal to the number of features specified. We use Topology Test B to extract a mesh with only one component.

The first step in our procedure is to compress the volume using the given isovalue to store only the voxels with neighbors on the other side of the isosurface in an octree (see Section 4.1.3). The first main loop is over the levels of this octree, beginning at the root. When we refer to a voxel at a given octree level, we really mean a supervoxel whose resolution is determined by the octree level. For example, at the root level, there is only one voxel. For each octree level we do four things—calculate distances (Section 4.1.4), initialize the boundary queues (Section 4.1.4), carve and add voxels starting at the boundaries (Algorithm 2), and subdivide the octree nodes for use in the next iteration (Section 4.1.3).

The important loop invariant for the octree level loop is as follows: At the beginning of the loop, any node that does not exist at the current level of the octree is missing because it would be a descendant of a node carved or added in an earlier iteration. This way we maintain the space savings of the octree by storing only voxels involved in the topology repair process.

Once the octree level loop is completed, the inside and outside sets each represent

Figure 21: Results for the Dragon model. On the left is the surface extracted from the original scan data, complete with holes from the undefined regions. Center is the genus-0 dragon with all handles (green) and holes (blue) patched. On the right is the result after opening the largest patch, a watertight mesh of genus 1.

a genus-0 version of the desired isosurface. If the isosurface is supposed to have higher genus, for each feature to open, one topology-altering operation is performed on the inside set and one on the outside set. In each case, the uncarved voxel v^* with the greatest distance is carved. Then the boundary queues are initialized with the uncarved neighbors of v^* and we again carve and add as much as possible. After each iteration of the loop, the genus will have increased by one and the handle formed will be the largest (in the sense described in Section 4.1.4).

After all the desired features are opened, the volume data set is changed to reflect the results determined by the procedure. Then any voxels that were not part of the original data (the patches or cuts and undefined voxels) may be smoothed using a procedure that respects topology (see Section 5.3).

Finally, an isosurface is extracted using Marching Cubes. To resolve the ambiguities, we use the variant described by Bhaniramka et al. [6] that is consistent with our local topology checks (see Section 3.3). Since every voxel with a neighbor on the other side of the isosurface was stored in the octree, a watertight mesh can be extracted directly from the octree representation.

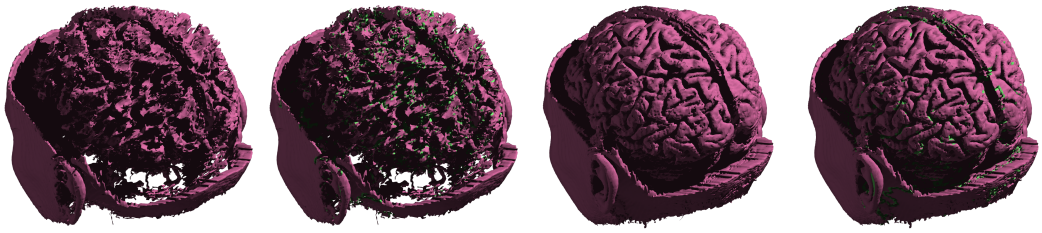
Figure 22: Results for David's head. Left is the result of surface extraction on the input data. There are many complex handles and holes in the mesh. Right is the result of surface extraction on the output of the process of carving from outside, with the option of leaving a layer uncarved at each step enabled.

Table 1: Running times for various models on a 2 GHz Pentium 4.

Data set	Voxels	Octree nodes 1	Octree nodes 2	Carving time 1	Carving time 2	Genus before	Genus after	Voxels changed
eight	39,865	22,233	45,785	0.4 s	0.7 s	2	0	320
brain 150	5,566,848	1,169,105	1,728,969	29.1 s	43.0 s	9,490	0	26,166
brain 175	5,566,848	1,593,585	2,104,233	41.4 s	52.9 s	18,364	0	58,756
Dragon low	9,228,168	2,059,513	3,127,689	60.4 s	85.2 s	44	1	347
Happy low	11,870,222	2,697,145	3,834,633	82.0 s	109.4 s	36	5	63
Dragon	68,458,320	14,952,729	19,439,177	492.0 s	611.4 s	93	1	2324
Happy	88,661,100	19,453,913	24,271,641	660.5 s	793.2 s	138	5	693
colon	121,372,672	59,384,913	66,346,489	16.4 m	19.8 m	80	0	104
David head	470,147,040	47,440,009	65,756,217	29.9 m	41.5 m	5342	0	29,279

4.1.6 Results

We used a 2 GHz Pentium 4 with 2 GB of RAM as our test system. Some results are summarized in Table 1. The times refer just to the carving and adding procedure (lines 2–15 of Algorithm 1). Time spent in the construction of the octree and the extraction of the output mesh are not included because these procedures are not a part of the topology simplification algorithm. The "eight" model is a volume containing the signed distance to the double torus depicted in Figure 15. The brain data set is an MRI scan from the Stanford volume data archive. The colon data is a CT scan obtained from Universität Tübingen's "Real World" medical data sets.



(a) (b) (c) (d)

Figure 23: Results for a brain scan at two different isovalues. The green triangles are from the voxels that were cut away to break the handles. a. Input volume for white matter (isovalue 150). b. Result of growing from inside. c. Input volume for gray matter (isovalue 175). d. Result of growing from inside.

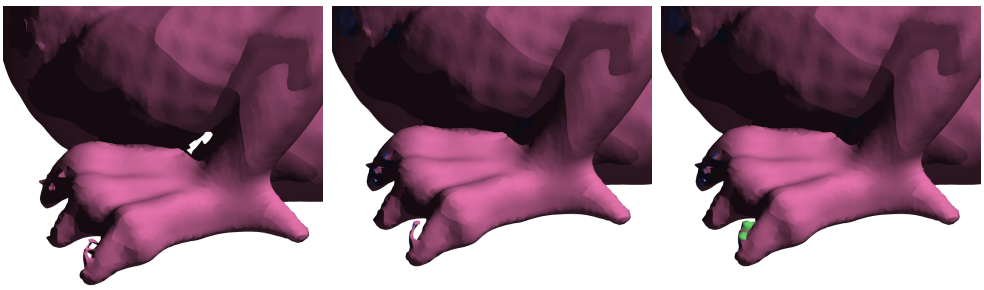


Figure 24: Results for the Dragon model, zoomed in on the left rear foot to show how topological noise is eliminated. On the left is the surface extracted from the original scan data. Center is the result from the procedure that grows from inside, and right is the result from the procedure that carves from outside.

The Dragon and the Happy Buddha data sets are partial volumes from the Stanford 3D Scanning Repository at the Stanford University Computer Graphics Laboratory. The David's head, also a partial volume, comes from the Digital Michelangelo Project, Stanford University. These three volumes were constructed from laser range scan data using Stanford's VRIP package [22]. The VRIP package takes as input multiple range images with alignment information and a desired resolution, and it provides as output a volumetric representation of the object ready for isosurface extraction. We used the volumetric representation to construct the octrees used in our procedure. For comparison purposes we used two versions each of the Dragon and Happy Buddha models—the highest resolution available in the scans and lower resolution models.

Figure 25: Results for a colon CT scan. Left: input volume. Right: result of growing from inside. Visually there is not much difference because the 80 repaired handles were small. The callout shows one handle cut by the procedure.

One can readily see the savings in space that the octree gives, especially for the larger models. Each node in the octree requires 13 bytes for storage—a 4-byte floating point value that holds the distance value used to determine the carving order as well as any subvoxel accuracy, a pointer to an array of children in the tree, a pointer from the parent, and a 1-byte field of flags used during the carving procedure. Without the octree compression, 5 bytes would be necessary per voxel to hold the floating point value and flags. Thus, for example, David's head would require 2242 MB of memory, but with the octree it requires only 590 MB.

The table shows two values for the number of octree nodes and the carving times. The first for each refers to the standard algorithm, and the second uses the option of leaving a layer of voxels uncarved at each resolution level. The extra layer uses more memory and more processing time to provide smoother results. The genus recorded before our procedure was run was the genus of the desired component only, since we saved only one component for each model.

There is one parameter the user must specify, and that is the genus of the output model. For example, the Dragon model has genus 1—the tail curves back and touches the body to form a handle. The input model has genus 44, as well as undefined regions that create boundaries on the resulting mesh. Our procedure determines a sign for every voxel, which results in patches for all the boundaries, and patches over all the handles. Then, since the user specified to open one handle, the largest handle is opened and we have a genus-1, watertight mesh that corresponds closely with the original data. In fact, to remove the topological noise we changed from outside to inside only 347 voxels. See Figure 21 for the Dragon model results.

4.2 Volumes

Rather than simplifying one one isosurface in a volume, it may be desirable to simplify all the isosurfaces in a cubical scalar field together. This is equivalent to reducing the number of critical points in the volume, since each critical point represents a topological feature in the volume. The idea is to construct a cubical scalar field with simple topology that is very similar to the original. This can then be used in any procedure that uses topological analysis. For example, as we have said contour trees [13] are often used as an aid to visualization or querying of contours in the volume. Reducing the number of critical points directly reduces the size of the contour tree. Therefore any process that uses the contour tree, whether a contour query algorithm or a user's exploration of the volume through contours, may be improved in terms of storage and access time.

We can extend the voxel carving technique described in the previous section to simplify all the isosurfaces in a cubical scalar field together [87]. The technique is the same as before—we start with the set of all voxels in a bounding box and carve them one at a time (while preserving topology), in order by decreasing value. The difference is we do not stop at a specific isovalue; instead we continue all the way

to the globally minimum voxel. In practical cases all voxels eventually get carved, and the voxel values are changed to reflect the carving order. The result is a cubical scalar field with one critical point, the global minimum.

In most cases a cubical scalar field will have some important topological features besides the global minimum. In this case the user may specify a threshold τ on the stability of the features. During the carving procedure, a voxel carve that causes a change in topology may be allowed, but only after the voxel has been delayed by an amount up to the value of the threshold. During the delay, the voxel may be able to be carved without changing topology because of carving of other voxels nearby. If the voxel does not get carved this way, once the delay has elapsed the voxel is carved and a topological feature is formed. This way some of the critical points of the cubical scalar field are retained, and since delaying did not help, those retained should be the most important critical points. We call these critical points *stable*.

The stable critical points are related to the points above the persistence threshold as defined by Edelsbrunner et al. [27]. The main difference between their work and ours is that they compute the global homology of the data set to determine which critical points represent the births and which represent the deaths of topological features. They delay only the death points so that corresponding birth points below the persistence threshold can be canceled. Since we do not compute any global topology, we cannot tell locally whether a critical point is a birth point or a death point, and therefore we must delay all critical points. We conjecture that this is the reason our algorithm may theoretically introduce more critical points during the carving process. In practice, however, this does not seem to be a problem (see Results section).

In detail, the carving procedure (Algorithm 3) is as follows: Initialize a priority queue to hold all the voxels in the bounding box, prioritized by highest value, and repeat until the queue is empty. Remove the top voxel v from the queue. If carving v

Algorithm 3 CarveAllVoxels

Require: A volume data set and the feature threshold τ
Ensure: The carving order

```
1: Insert all voxels into priority queue and mark as queued
2: while priority queue is not empty do
3:    Remove v from queue
4:    Mark v as not queued
5:    if v is not already carved then
6:       if OKToCarve(v) then
7:          Carve v
8:          for all n such that n is a 26-neighbor of v do
9:             if n is not already carved and not marked as queued then
10:               Add n to the priority queue
11:               Mark n as queued
12:      else
13:         Add v to the priority queue with priority decreased by τ.
```

Algorithm 4 OKToCarve

Require: A voxel v and the current state of the carving process
Ensure: Whether v should be carved at this point

```
1: if carving v does not change topology then
2:    return true
3: else if v has already been delayed then
4:    return true
5: else
6:    return false
```

does not cause a topology change, go ahead and carve it and add its neighbors to the queue if they are not already there. If carving v would cause a topology change and v has not already been delayed, push v back onto the priority queue with its value decreased by τ. If v has already been delayed, carve v and add its neighbors to the queue that are not already there. The neighbors are pushed onto the queue when a voxel is carved because a neighbor that was delayed earlier because it was a topology change may not be a topology change now that v is carved. Once this procedure is complete and all the voxels are carved, the voxel values are changed to reflect the carving order. Note that each voxel has value within τ of its original value.

In 2D, we can describe the effect of this algorithm as follows: Consider the cubical scalar field as a height function. If we start at the outside and carve voxels, the procedure flattens out each local maximum until it is level with the nearest saddle point (or the threshold is reached) and digs trenches to connect saddle points to local minima. The complementary actions occur if the algorithm starts from the global minimum and adds voxels. In this case local minima are filled in to the nearest saddle point, and ridges are built up on saddle points to the nearest local maximum. See Figure 26. In 3D the analogous changes are made to the volume.

Algorithm 5 OKToCarveSecondary

Require: A voxel v, the current state of the carving process, and the previous carving order
Ensure: Whether v should be carved at this point
1: **if** carving v does not change topology **then**
2: **return true**
3: **else if** v has already been delayed **then**
4: **return true**
5: **else if** v was a stable critical point in the previous pass **and** all neighbors of v carved before v in the previous pass have already been carved in this pass **then**
6: **return true**
7: **else**
8: **return false**

Because the stable topology changes are flattened out to the point of the threshold,

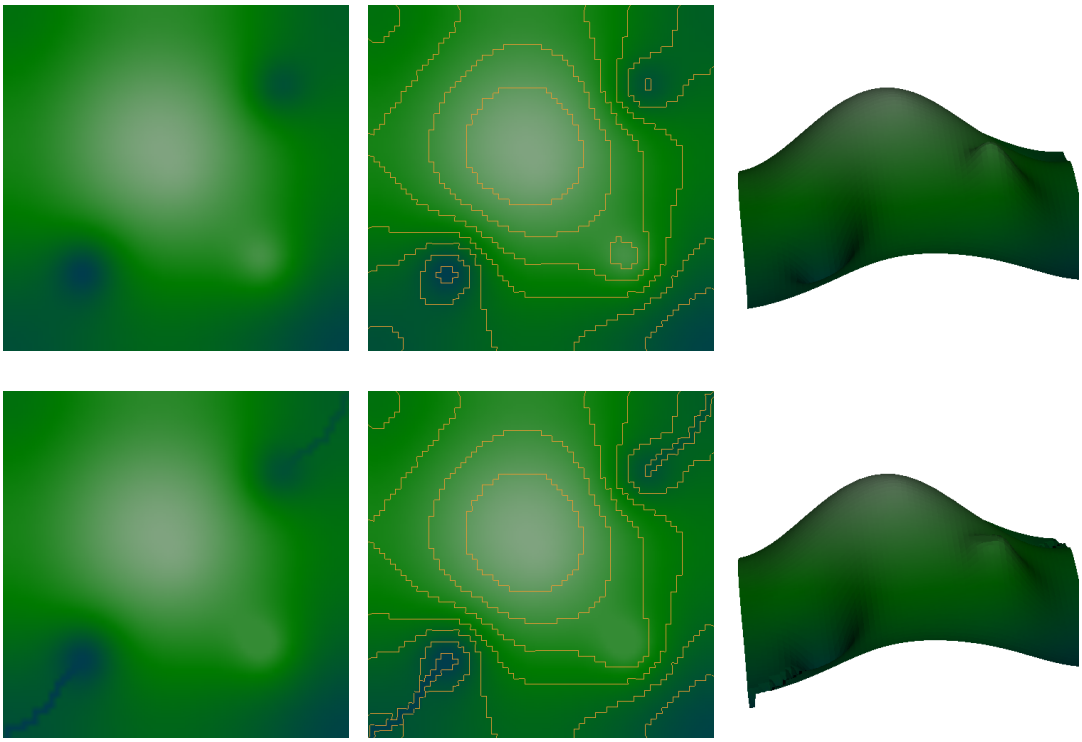


Figure 26: A 2D scalar field. White represents high values and blue represents low values. Top is the input and bottom is the result of carving from inside starting at the global max. In this case we assume everything outside the bounding box is very low, so the results are complementary to what the text describes. This example is like the top of a mountain with two peaks and two small valleys. The topology simplification removes all critical points except the global max. Note that the local max is leveled out and the local mins are connected to the global min outside the bounding box.

the results of the algorithm are not as close to the original volume as they could be, though the maximum difference is of course τ. See Figure 27. However, now that we know which critical points are the stable ones, a second pass may be performed. The algorithm is the same as the above, except when the voxel v at the top of the queue is known to be critical from the first pass. If all of v's neighbors that were carved before v in the first pass have already been carved, then we go ahead and carve v. Otherwise we delay it as usual. This check of the neighbors is necessary for complex critical points that may have a stable and a non-stable component (see Figure 28 for a 2D example). Algorithm 5 shows the replacement check for line 6 of Algorithm 3.

The purpose of the secondary passes in the multiple-pass algorithm is to make

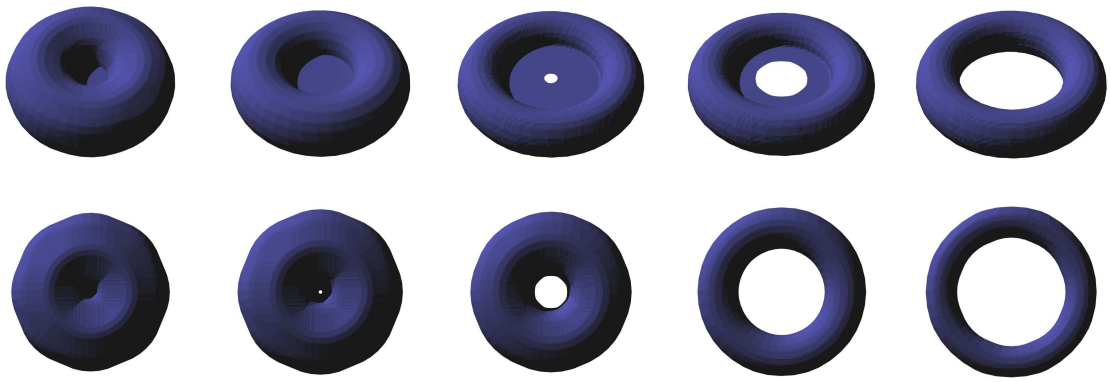


Figure 27: Snapshots of the carving process for the single-pass algorithm (top row) and the second pass of the multiple-pass algorithm (bottom row) for a signed distance function for a torus. Both procedures require the same number of topology changes. The carving order produced by the multiple-pass algorithm much better reflects the ordering of voxels by value.

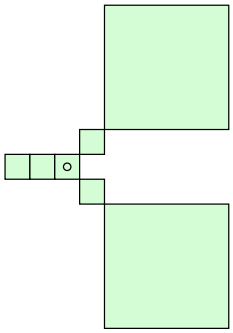


Figure 28: Delay of a complex critical point. The marked voxel is being considered for carving during the second pass. In the first pass it was marked stable because carving it disconnects two large components. In the second pass it must be delayed, however, or an unstable critical point from the small component will be introduced.

the output volume closer to the original volume, in terms of the number of voxels changed. In fact, just a single second pass is enough in most cases. In principle the secondary passes may introduce new critical points, but in practice this does not happen too much. Algorithm 6 shows the whole multiple-pass volume topology repair procedure. Note that in the output volume, the voxel values decrease monotonically when viewed in the carving order. The use of the multiple-pass algorithm rather than the single-pass algorithm is analogous to Edelsbrunner et al.'s use of square persistence diagrams rather than triangles [27], since the purpose of the squares is to

Algorithm 6 FixVolumeTopology

Require: A volume data set, the feature threshold τ, and the number of passes
Ensure: A volume data set within τ of the original

```
1: CarveAllVoxels
2: for pass = 2 to the number of passes do
3:    CarveAllVoxelsSecondary
4: Initialize the current value c = ∞.
5: for all voxels v in order according to carving order of last pass do
6:    if the value of v is less than c then
7:       Set output value of v to input value of v
8:       Set c to the value of v
9:    else
10:      Set the output value of v to c
```

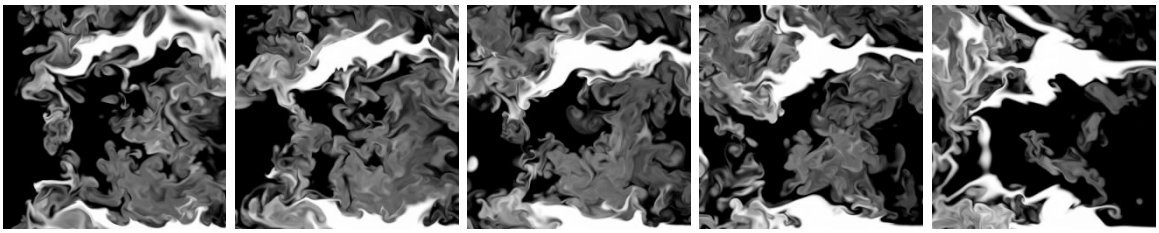


Figure 29: Slices through the fluid simulation data set.

leave the persistent critical points intact rather than decreasing their persistence.

4.2.1 Results

We have tested our algorithm for a number of volume data sets. Below, we focus our attention on three representative data sets: an approximate signed distance function from the well-known the Buddha model [22] sampled on a $181 \times 181 \times 434$ grid (it has positive values inside the Buddha and negative values outside), a $256 \times 256 \times 128$ time slice of a fluid dynamics simulation and a $241 \times 281 \times 191$ subvolume of a chest

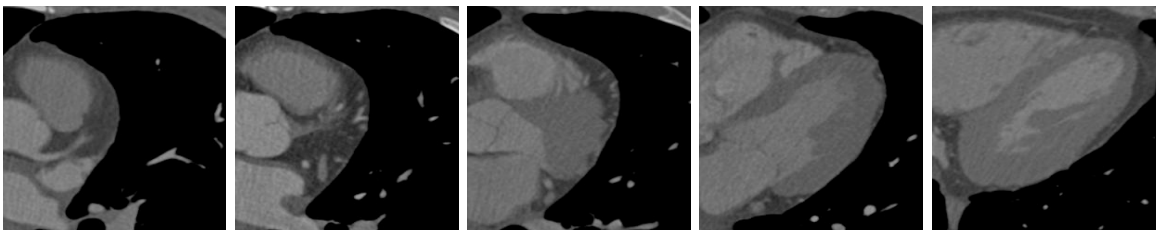


Figure 30: Slices through the CT scan data set.

Threshold	original	1	3	5	10	15	20	30
Critical Points	23832	209	95	71	43	25	17	3
CT nodes	12531	82	40	30	16	10	6	2
β_0	7	1	1	1	1	1	1	1
β_1	22	14	12	12	10	6	4	0
Altered voxels								
(1 pass)	N/A	20244	40809	65750	152173	241968	353392	417176
(2 passes)	N/A	16477	22030	24310	98853	134575	195835	417135

Table 2: Results for the signed distance volume for the Buddha model. We used 0 as the isovalue for isosurfaces whose Betti numbers are listed in the table. The isosurfaces were always closed (therefore $\beta_2 = \beta_0$).

Threshold	original	0.5	1.5	2.5	4.5
Critical points	321483	198467	118826	82818	47671
CT nodes	130425	63706	32708	20718	10796
β_0	507	507	423	362	260
β_1	2786	2786	2644	2450	2079
β_2	402	402	327	269	178
Altered voxels					
(1 pass)	N/A	0	241597	555765	611270
(2 passes)	N/A	0	232670	554158	608357

8.5	16.5	32.5	64.5	128.5	256.5
21873	7450	1988	482	56	2
4608	1814	574	152	24	2
182	124	60	17	2	1
1500	811	180	38	2	0
111	67	24	5	0	0
695387	809054	950320	1069321	1134626	1129825
693583	800258	931539	1037439	1127575	1129825

Table 3: Results for fluid simulation data set (Betti numbers shown correspond to isovalue 127.5). Note that the intensities of all voxels are integers and therefore only infinitesimal perturbation is performed for $\tau = 0.5$—this is why the reported count of altered voxels is zero.

CT scan. The data sets vary substantially in character: the first two are synthetic, the last one is acquired. Synthetic data sets are known to be easier for topological algorithms as they contain less topological noise [13]. The signed distance function is relatively simple while the fluid simulation data set is quite complex (Figure 29). The CT scan volume, like most acquired data sets, is noisy: it contains a high number of low-stability critical points. Slices through the CT scan volume are shown in Figure 30.

First, we compare several performance characteristics of our algorithm for each of the three data sets and for a variety of stability thresholds:

(a) The number of critical points in the output data set

Threshold	original	0.05	0.1	0.2
Critical points	584879	46346	7780	1395
CT nodes	274338	24953	4713	853
β_0	1081	417	245	82
β_1	3339	997	381	109
β_2	886	307	165	41
Altered voxels				
(1 pass)	N/A	1172476	1432351	1513028
(2 passes)	N/A	1157489	1408416	1561595

0.6	1.0	1.4	1.8	2.0
166	92	53	10	2
65	32	16	6	2
14	7	6	3	1
25	2	1	0	0
2	1	0	0	0
3171618	3878776	4481524	4487567	4487685
3148997	3861051	4450089	4482140	4487685

Table 4: Results for the chest CT scan; Betti numbers of isosurface for isovalue of 0.7 are shown.

(b) The number of contour tree nodes in the output data set (after removing all regular nodes, i.e. nodes having one neighbor above and one neighbor below)

(c) The Betti numbers of the isosurface corresponding to an arbitrarily selected isovalue

(d) The number of voxels in the output volume that have a different value than in the input volume (we think of it as a measure of the amount of change applied to the data); we compare these numbers for the one-pass and two-pass version of our algorithm. When comparing the voxel values, we disregard the infinitesimal perturbation.

The results for each of the three models are shown in Tables 2, 3 and 4. The tables show that our procedure can highly reduce the number of critical points in a volume data set and the size of its contour tree. In particular, for all data sets the maximum simplification has a small number of critical points (2 or 3). This is true for most data sets we experimented with, except for those described in Section 7.2.2. The tables also demonstrate that the isosurfaces in the output volume are also topologically simplified. Finally, the signed distance data sets demonstrate the usefulness of the

Figure 31: Isosurfaces of the Buddha model corresponding to the isovalue of zero for thresholds 1, 10, 15, 20 and 30.

Figure 32: Snapshots of the carving procedure for $\tau = 20$. Note that all surfaces throughout the process are connected and have genus between 0 and 2.

second pass: the two-pass version of the algorithm alters fewer voxel values. The difference is much smaller for other, less regular, data sets.

The algorithm described in this section can be thought of as a variation of our procedure [84] (described in Section 4.1) that simplifies all isosurfaces rather than just one. Isosurfaces corresponding to isovalue of 0 for different simplified versions of the signed distance data set are shown in Figure 31. Snapshots of the carving process (or, equivalently, isosurfaces in the output volume corresponding to different isovalues) are shown in Figure 32. Contour trees for the topologically simplified CT scan (rendered using graphviz [37, 38]) are shown in Figure 33.

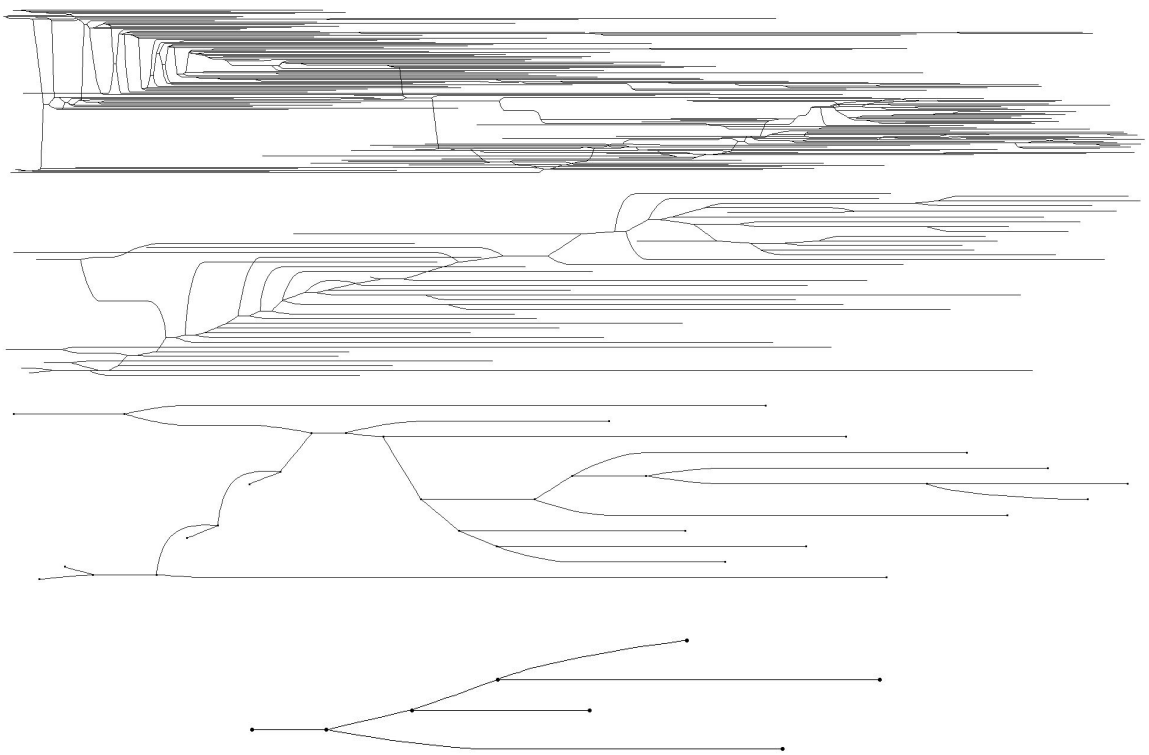


Figure 33: Contour trees for the topologically simplified chest CT scan (thresholds: 0.3, 0.5, 1.0, 1.6).

4.2.2 Comparison to topology-insensitive filters

For comparison purposes, we have performed some simple experiments using filtering methods that ignore local and global topology to clean up global topology. See Figure 34 for results on a subsampled version of the happy Buddha model. We compared the results of carving with the results of a low-pass filter. The genus of both results was the same, four. The carving procedure started from the inside and delayed voxels that change topology. The handles at the top were broken because the handles there are thin; a smaller threshold would recover them correctly instead of breaking them. We used a very simple low-pass filter on the volume, the average of the voxel and its 6 face neighbors. The filtering result broke the handles in the same place as the carving procedure but left behind some floating components. There are six small, unimportant handles in this data set that were removed by both methods. They were

all cut using the carving method, but it is difficult to say what happened to them in the filtering method. It appears that they were parts of thin geometric features that were blurred away.

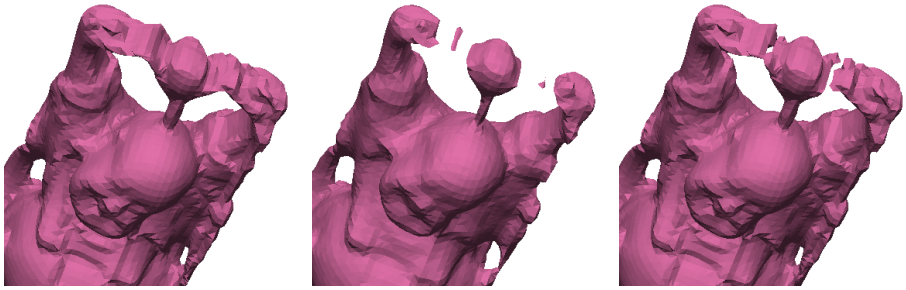


Figure 34: Results of filtering and carving, isovalue 0 of the signed distance to the Buddha model. Left is the input, center is the result of low-pass filtering, and right is the result of carving.

The low-pass filter result is not acceptable because one of the goals of this thesis is to leave regions of the volume not involved with topology simplification intact. The low-pass filter simplifies the topology but also blurs the geometry throughout the volume.

CHAPTER 5

PARTIALLY-DEFINED VOLUMES

An interesting aspect of our technique is that it may be used in procedures that repair volumetric data sets with undefined regions.

Definition A *partially-defined* cubical scalar field is a function h that assigns a scalar value to the center of each d-dimensional cell of a subset of a regular rectangular lattice of R^d.

The goal is to simplify the topology of such data sets and extract watertight isosurfaces with simple topology, while at the same we wish to retain the original geometry in areas not associated with the simplification.

5.1 Polygonal Data

Sometimes polygonal models are provided that are incomplete. This incompleteness can be in the form of missing triangles, so that the mesh has boundary (holes). On the other hand, the model may be a polygon soup where there is no connectivity information at all. In situations where the original connectivity is not important, a valid way to process the mesh is to convert it to a voxel format, process it, and extract a surface using marching cubes or some other isosurface algorithm.

Much work has already used this idea for mesh repair and simplification. Nooruddin and Turk [69] repair holes in a mesh by first "voxelizing" the mesh. Their volumetric process consists of morphological operators used to smooth out problems such as small holes and cracks. This does repair some topological noise as well, but fine geometric features are also eliminated. Their method is intended mainly for repair of problematic polygonal meshes, but their methods for voxelization of polygonal data

could be used as a starting point for our algorithm.

He et al. [49] also present a method for combined topology and geometry simplification. Their method uses several low-pass filters on a volume converted from an input polygonal model to generate models of varying detail. The topology is simplified but at the cost of eliminating other high-frequency information.

Ju, in [56], describes a volumetric method for mesh repair. This method stores the model in an octree in order to save space, as we do. The goal is a subset of ours—to create a watertight mesh. The divide-and-conquer technique for patching holes is nice because it is simple and in many cases fast, but the types of patches that can be constructed is limited by the assumption that all holes have a boundary of a single closed loop.

All of these methods [69, 49, 56] simplify the geometry and topology of the models, but our purpose is to simplify the topology only, while leaving the geometry of the original model intact (in regions not associated with the topology simplifications). We have used our voxel-carving algorithm [84] to repair the topology of watertight triangle meshes by first scan-converting the triangles into a volumetric representation at a resolution given by the user. A signed-distance field is constructed to order the carving algorithm. In principle it could be used with any of the above repair procedures to construct a volume with controlled topology.

5.2 Isosurfaces from Range Scans

Laser range scan subjects typically have areas of the surface that the laser and camera cannot reach, such as deep concavities and other regions that are obstructed from any accessible view. When volumetric techniques are used to align scans, the unseen areas leave behind voxels with no defined sense of inside and outside. These undefined regions leave holes (regions of missing triangles) in the extracted isosurface unless handled somehow.

Curless and Levoy [22] describe how to align range images from laser range scans using a volumetric technique. They address the issue of unseen regions using a method called "space carving." This method uses line-of-sight information from the scan to carve out regions of space where the model cannot be. This gives a first guess at the undefined regions of the volume but still cannot accommodate highly convex areas.

Davis et al. [23] build on Curless and Levoy's technique using a method called "volumetric diffusion" to patch the holes resulting from the undefined regions. They define a distance function over the undefined regions based on nearby defined voxels and combine it with a smoothing filter to create nice-looking patches over the holes in the extracted mesh. This would make an excellent preprocess or concurrent process to our topology repair process, since we make no attempts to smooth the geometry, and they make no attempts to control the topology.

Volumetric diffusion has nice results but takes extra work. Space carving is a good start but still leaves some areas undefined. Our voxel carving and adding procedures (see Section 4.1), when used concurrently, can be used to fill in undefined regions completely with no extra work. The process that carves from outside stops when it hits the object surface, but it also must stop when it meets the set that grows from inside at undefined regions. Likewise the set that grows from inside stops when it hits the object surface or when it hits the boundary of the uncarved set of the outside procedure.

When a voxel is carved it is marked outside, and when a voxel is added it is marked inside. The result is a completely defined sense of inside and outside for every voxel and therefore a watertight extracted mesh. Pairs of voxels that were originally undefined but are marked on opposite sides after the procedure result in patches in the extracted isosurface. These patches may not be very high quality or even minimal in any sense, but they are obtained simply and automatically. We mark the voxels that make up the patches for postprocessing (see Section 5.3) if the user

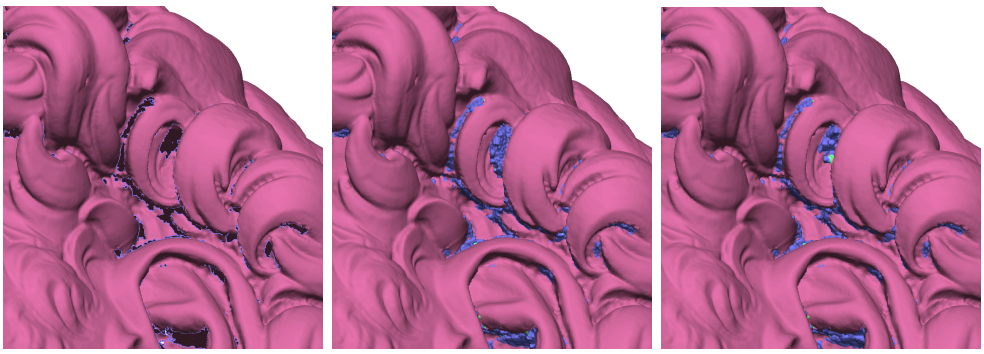


Figure 35: A close up of David's head just above the left ear. On the left is the input data; the boundary edges are highlighted. Center is the output with the uncarving option off and right is the output with uncarving on. Blue triangles came from areas where the surface from inside met the surface from outside during the carving and growing processes (patches over holes). Green triangles came from voxels where the surface from outside stopped carving due to failing the topology test (patches across handles).

desires, since they were not part of the original model. See Figure 35 for a close look at some of the holes in David's hair.

There is a technical detail in this method of dealing with the holes caused by undefined voxel regions. If the carving process cannot approach the surface of the object near an undefined region (because, for example, a patch is being formed across a handle), the inside set is free to leak out the hole (see Figure 36). This is a common problem because very often patches are formed in the same highly concave areas of the object that the laser range scanner cannot see (such as in the Dragon's tail when it loops back to form a handle). To prevent these leaks, when a voxel is carved from outside, its uncarved undefined neighboring voxels are marked "unaddable" to tell the growing process not to add them to the inside set. Likewise when a voxel is added to the inside set, its undefined neighbors are marked "uncarveable" so that they cannot be carved from the outside. This way, while a handle patch (or cut) is being formed, even though the voxels are not carved and may therefore still be undefined, the marks prevent the inside set from leaking into the uncarved region.

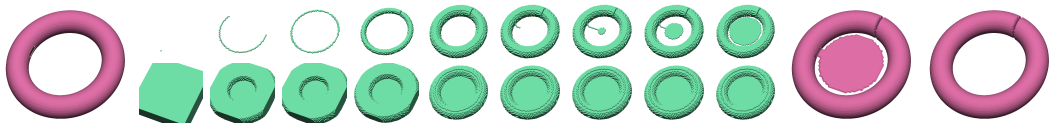


Figure 36: Leaking through an unknown region. Left is the isosurface of the input data; it has a hole. Center top is the progress of adding from the inside and center bottom is the progress of carving from the outside. In the sixth frame the inside set has begun to leak out the hole into the uncarved voxels forming the patch. Right is the final result and far right is the desired result, both from the inside set.

5.3 Smoothing

Our procedure patches holes automatically, but the results are not very smooth because we only use one or two layers of voxels on each side of the isosurface. It is possible to improve the shape of the patches further (both across the handles and over undefined regions) with postprocessing. Our algorithm identifies certain voxels that have a newly-defined sense of inside and outside. These are the voxels that make up the handle patches or cuts, which are changed from outside to inside or vice versa, as well as the voxels in formerly undefined regions that now are defined to lie on the surface of the object.

One possibility for improving the patches is to do some global smoothing, such as a gaussian filter, over all the voxels that were changed. We have implemented a simple smoothing procedure that repeatedly applies a gaussian filter over a $3 \times 3 \times 3$ window. Voxels from the original volume data are left alone, and other voxels are set to a weighted average of their neighbors. Voxels are allowed to change from inside to outside only if the change does not cause a change in topology. This is very important because otherwise the results of our topological simplification may be corrupted. The same local check for topology change during carving can be used during smoothing. Initial results look good (see Figure 37), but more work is needed to integrate this with our technique more fully and to give the patches better quality. The basic goal would be to make the patches look as much like the surrounding mesh as possible, as others have explained [77, 83]. Smoothing could also be done in the mesh domain

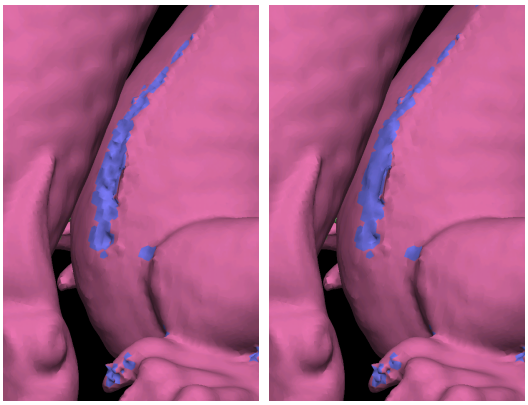


Figure 37: Smoothing the patches over holes in the Dragon model. Each figure is the result of running marching cubes on the output volume. On the right, before running marching cubes, the gaussian filter was run on the repaired volume.

rather than the volume domain, although the topological guarantees would not be preserved (one would have to be sure that the mesh does not intersect itself during the smoothing process).

5.4 Conclusions

Mesh repair is clearly a related application to our procedure. It is possible that the object whose topology the user wishes to control is originally defined by a polygon soup—a list of polygons with no connectivity information. The polygons may overlap or have gaps between them. As long as there is some sense of inside and outside—such as polygon normal vectors—the set of polygons could easily be scan-converted into a volume grid. If the gaps are not too big, then the surface carved from the outside should meet the surface grown from the inside near the scan-converted polygons. A new connected, manifold mesh could then be extracted using marching cubes.

On the other hand, if the object the user wishes to simplify is given as a mesh with connectivity information, our current procedure has no method of retaining the original connectivity while simplifying only the regions with topological noise. This would probably require some sort of polygon stitching, and that is outside the scope of this technique as we have applied it at this time.

CHAPTER 6

TIME-DEPENDENT DATA

Users are no longer content to use static data only—fluid and particle simulations are just one example of time-varying volume data [65]. However, volumetric data sets are huge, especially after adding a dimension for time. As we stated in the introduction, what is needed is a small topological representation that can be used to learn about and discover properties of the data set. In particular, it would be very useful to find isovalues and locations in the data according to user-specified parameters, such as volume, genus, and life span over time. In general, the representation should depict the contour dynamics and should support queries of individual contours as well as statistics on the contours overall.

6.1 Topological representation

The contour tree (see Section 3.4) is a very helpful representation for these sorts of queries in volumetric data, and the contour tree can be constructed for data of any dimension [14]. However, treating a 2D plus time or 3D plus time data set as a 3D or 4D data set (respectively) throws away some important information, namely that the time dimension is special. In fact, in some ways the time dimension is more like the isovalue than the other dimensions of the space the cubical scalar field is defined on. We are interested in how the topology of a level set evolves over time and as the isovalue varies, rather than the topology of the level set in the higher-dimensional space. Therefore, we consider the data on a slice-by-slice basis but also include information relating adjacent slices. The slicing can be with respect to time or some other parameter.

Definition A cubical scalar field on a lattice of R^{d+1} is *parameter-dependent* when one dimension of the domain is singled out as a parameter. The domain is rewritten $R^d \times R$.

Definition The ith *slice* of a parameter-dependent cubical scalar field h (defined on $R^d \times R$) is a cubical scalar field h_i (on R^d) defined by fixing the parameter to i.

Definition A *window* from i to j ($i < j$) is a subfield of a parameter-dependent cubical scalar field defined on the sublattice $R^d \times [i, j]$. The shortest possible window ($j = i + 1$) is called a *thick slice*.

When we analyzed volume data that was not parameter-dependent, we assumed all cubical scalar fields were defined on all of R^d. For real-world, finite data sets contained in a bounding box, we extended them to the full space by assuming that all voxels outside the bounding box were greater (or less) in value than all the voxels inside the bounding box and contained no critical points. The purpose of this was convenience of implementation because it allowed us to ignore the case of topology changes on the boundary of the data set. The assumption seems justified for many cases such as CT scans and laser range scans because outside of some region everything can be considered empty space.

However, this assumption is not justified in the time dimension or on window boundaries, because typically you would not assume that everything just disappeared at the next time step outside the window. Furthermore we need to keep information about changes on the boundary to use when merging thick slices. Therefore we use a separate definition of criticality for voxels on the parameter window boundary. To test whether carving or adding a boundary voxel v will change topology, we use the same check as for an interior voxel with the missing neighbors filled in as follows. If the topology-preservation test passes when the missing neighbors are assumed below

v and the topology-preservation test passes when the missing neighbors are assumed above v, then v passes the topology preservation test.

We use thick slices and the subdomain-aware contour trees of Szymczak [85] to represent the relation of adjacent slices in our topological representation.

Definition The *subdomain-aware contour tree* of a domain X and subdomain $Y \subseteq X$ is the contour tree of X augmented with the critical points of Y. Each tree edge e is labeled with the number of contours of Y that correspond to e. The correspondence is given by the inclusion-induced map described in Section 3.4.4. Note that join trees and split trees can be made subdomain-aware as well.

The data representation consists of a contour tree of each slice of the data set and a subdomain-aware contour tree of each thick slice. We combine this with maps from the slice (the subdomain) contour trees into the thick slice contour trees. These maps are induced from the inclusion maps from the slices into the thick slices. The contour trees of the thick slices can be combined to form as much of the full-space contour tree as desired using the recursive approach described by Pascucci [71] (see Section 3.4.5).

6.2 Sliced spatial data

When viewing 3D medical images such as CT and MRI scans, specialists often step through one slice at a time in order to see all the information provided by the scans. Motivated by this parameterization of the scalar field, we have implemented a technique for segmentation of airways in the lungs. The technique uses contour trees of windows of a fixed size and inclusion-induced maps of contour trees of slices corresponding to the window boundaries. The assumption is that branching tubular structures roughly perpendicular to the slices will have one component at one end of the window and between one and three components at the other end.

The algorithm is as follows. First the join trees for all thick slices are constructed and merged to form the join trees for the windows. Enough windows are used to cover all of the data; for our experiments we used windows of the form $[i, j]$ where $j = i + 10$ and i is a multiple of 5. Each window has two boundary slices, so we constructed the join trees of these as well and used the inclusion-induced map to count the components that each sublevel set component C of the window w formed when it overlapped the slice. We used sublevel sets (and the join tree) because the airways have low value, so anything below a certain threshold can be considered air. If C has between one and three components in each slice, then it is likely to be a possibly branching tubular structure. This is the only assumption on the shape of the sublevel sets that we have. All sublevel sets below the vessel wall/air threshold that meet these criteria are selected, and then sublevel sets contained in other selected sets are thrown out since they are subsumed by the containing sets.

We then use the contour trees of the windows and the algorithm described in Section 3.4.1 to compute the Betti numbers of the sublevel sets. Since the sublevel sets are supposed to be airways they should have 0 handles. One cavity is allowed in case of occurrence of one bright voxel inside the tube (due to noise). The rest of the sublevel sets are thrown out. Finally, all the voxels from the selected level sets are selected.

This procedure is used separately three times, once with each dimension used as the slicing parameter. The union of all the selected voxels is the final result. See Figure 38.

Initial results are at least as good as extracting an isosurface with a constant isovalue and in some cases better. More could probably be done by more advanced shape analysis, since the false positives are not very tube-like in terms of shape. However, we wanted to limit our analysis to topology to see how well such tests could perform.

Figure 38: Bronchi segmentation, the output of our slicing procedure. Left to right are three different data sets.

6.3 Topology simplification

Tracking contours through time is an interesting problem. Szymczak [85] has done this using subdomain-aware contour trees. We can speed up and improve the process by simplifying the data first to reduce the size of the contour trees. The topology simplification algorithm should first of all simplify the topology of data of any dimensionality, while leaving intact the regions not associated with the simplifications. Second, it should reduce the size of the topological representation so that queries (such as contour tracking) will be faster. Previous approaches [15, 44] give no indication how to extend their techniques to time-dependent data, but we have extended our topology simplification technique to time-dependent data by changing the local test for topology change. Furthermore, our technique edits the volume directly and only edits the volume to cancel critical point pairs, so the rest of the volume is left untouched. Since the number of critical points is reduced, the size of the representation (the subdomain-aware contour trees) is reduced. Therefore our technique fulfills all the requirements for the needed topology simplification algorithm.

We have extended our techniques for volume topology simplification using voxel carving (Section 4.2) to time-dependent volume data sets. There are two obvious ways to do this that are not sufficient. The first obvious way is to process each time slice individually using our standard topology repair technique. This is not sufficient,

however, because there will not necessarily be any continuity between the repairs in one slice with the repairs in the next. For example, a break in a handle could be in a very different place with slightly different initial conditions, and these conditions may vary slightly from one slice to the next as the object evolves in time.

The next obvious possibility is to consider time as another spatial dimension. Our standard technique of processing one voxel at a time using a local check for topology change generalizes immediately to higher dimensions. This is also the most obvious way to extend other simplification techniques such as Carr's [15] to time-varying data. However, if time is simply treated as a third or fourth spatial dimension, some information is probably lost because time is special. Consider the following two-dimensional (plus time) example: There are two discs in the plane; over time they approach one another and then merge. If time is thought of as a spatial dimension, in 3D this looks something like a solid letter V. If we simplify the topology in 3D, nothing is changed because it is already a topological sphere. On the contrary what we desire is this: each slice should be a topological disc (the two discs with a thread between them), but the resulting animation should still be continuous. See Figures 39 and 40.

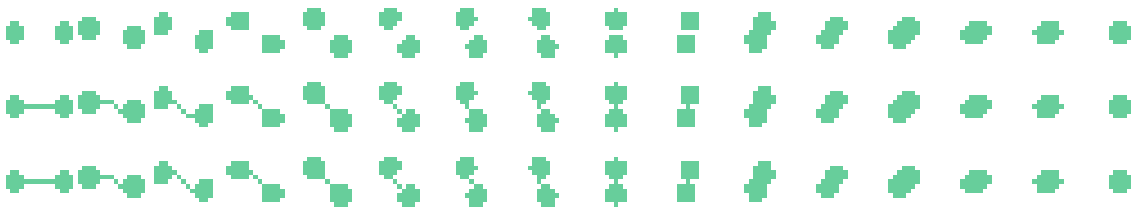


Figure 39: Two discs in the plane shown as 16 time slices. Top is the input volume. It also reflects what the output looks like when carving with the fully 3D topology check. Center is the result of carving slice-by-slice, and bottom is the desired output—the result of carving using the combo check. Note particularly slices 3 and 7—the combo check gives results that are more similar to neighboring slices.

Therefore, we use a special local topology test for time-dependent data. The voxels are processed in the full space, including the time dimension. However, a voxel

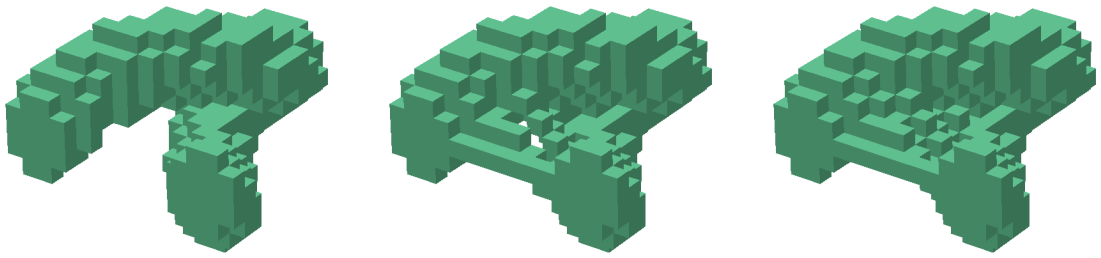


Figure 40: Two discs in the plane shown as a volume in 3D. Left is the input volume. Center is the result of carving slice-by-slice. The visible holes demonstrate the discontinuity of the slice-by-slice method. Right is the desired output—the result of carving using the combo check.

is considered to change the topology if it changes the topology in the full space or if it changes the topology in its time slice. Thus this voxel is safe to carve only when it does not change topology in either sense. This treats time separately but also maintains a connection between adjacent time slices, and therefore it should prevent both problems described above. We call this the "combination topology check," or "combo check" for short.

We have implemented topology simplification using the combo check and run it on time-varying two-dimensional and three-dimensional data. We use the number of nodes of the thick-slice contour trees as a measure of success of the topology simplification, since it corresponds to the size of our time-varying topological representation. The size of the set of subdomain-aware contour trees is the sum of the numbers of critical points in the thick slices. Section 6.4 shows results in two dimensions plus time and Section 6.5 shows results for three dimensions plus time.

6.4 Two dimensions plus time

For an example of a time-varying two dimensional data set we use a simulation, the spiral wave. The data set consists of 50 slices of 128×128 voxels. The voxel values

Table 5: Topological complexity and running times for spiral wave data set on a 2 GHz Pentium 4. Threshold value is 10 (3.9% of the height field range). The number of voxels is $128 \times 128 \times 50 = 819,200$.

Topology check	Critical before	Critical after	Time (s)
slices	51964	15850	10.0
fully 3D	51964	11860	12.3
combo	51964	1502	15.0

are in the range $[0, 255]$. The original data set has 51964 total nodes in the thick slice contour trees. The large number is partially due to nearly flat regions. After topology simplification using the combination check, with a threshold of 10, the total number of nodes is only 1502. See Table 5 for comparison of the three different topology checks—slice-by-slice, fully 3D (ignoring that time is special), and the combination check. It is clear that the combination topology check yields the best savings in contour tree size.

We have used the results of topology simplification for contour tracking in the spiral set. The topology simplification allows tracking of contours (and sublevel sets) at higher isovalues than would work without topology simplification. See Figure 41. The sublevel set of interest is highlighted in red (in the top row). If the isovalue is increased (middle row) the sublevel set is lost when it merges with its neighbors. After topology simplification (bottom row), the region of interest remains separate at the higher isovalue for several time slices. This is beneficial because the tracked voxel set is bigger and therefore may contain more useful information. The topology simplification allows the separations between components to be preserved as the isovalue increases, until the threshold is reached. Simply put, it allows the power of a higher isovalue with the control of a lower isovalue for contour tracking.

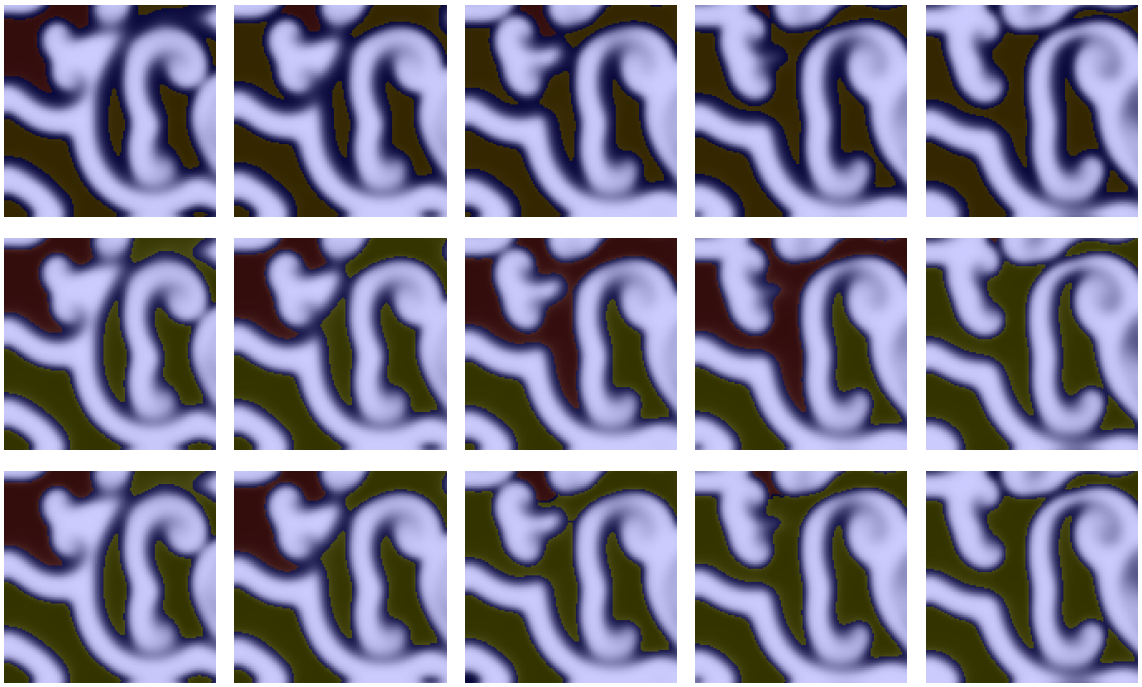


Figure 41: An example of sublevel set tracking in the spiral data set. Time increases from left to right. Brown is below the isovalue, blue and white are above the isovalue, and red is the tracked sublevel set. Top row shows tracking without topology simplification at a low isovalue (13). Middle is at a higher isovalue (32) without topology simplification. Bottom row is with simplification with threshold 25 at the higher isovalue (32). The third and fourth time slices show the most difference.

6.5 Three dimensions plus time

We have also implemented our simplification algorithm in four dimensions, using the combination topology check that checks the local neighborhood for topology changes in 3D in the slice and in 4D. See Table 6 and Table 7 for comparisons of the three different topology checks. The bubbles data set [60] (Figure 42) is a simulation of air bubbles moving, joining, and splitting in water. The volume is a signed distance field to the air-water boundary. The voxels of the fluid simulation data set (Figure 43) are samples of the density of air during an explosion.

The results of the combo check in three dimensions plus time for the fluid simulation show only a small improvement (at best) over the slice-by-slice approach. However, the improvement of the combo check over the slice-by-slice approach on

Table 6: Topological complexity and running times for bubbles data set on a 2 GHz Pentium 4. Threshold value is 14 (10% of the height field range). The number of voxels is $84 \times 84 \times 84 \times 50 = 29,635,200$.

		From inside		From outside	
Topology check	Critical before	Critical after	Time (min)	Critical after	Time (min)
slices	1047900	76885	5:03	59121	4:45
fully 4D	1047900	819638	15:12	830418	14:40
combo	1047900	208	24:47	4620	15:12

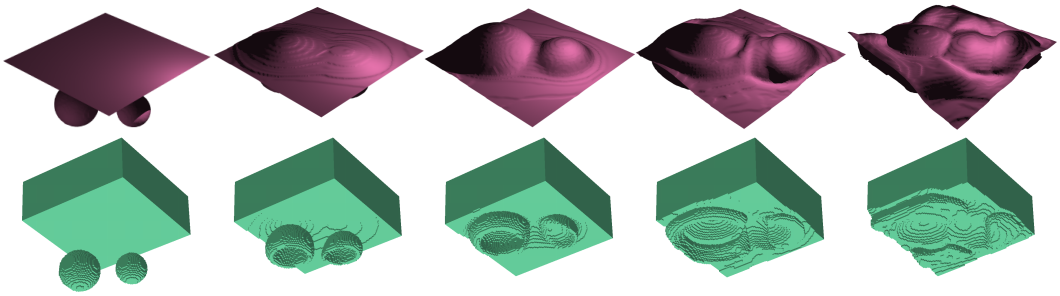


Figure 42: Some time slices of the bubbles data set. Top row shows the surface of the water from above. Bottom row shows the air bubbles from below.

the bubbles data set is dramatic. One possible reason for this is the following. The bubbles data set is a distance field, and there are sets of voxels that are equidistant from the air-water boundary. If there is a critical voxel in one of these sets, there may be many possible locations for it in each slice, and where exactly it will occur depends on the carving order. Since the combo check enforces continuity between slices, the critical points in adjacent slices will be adjacent, and thus the number of thick slice critical points will be smaller than the arbitrary locations determined by

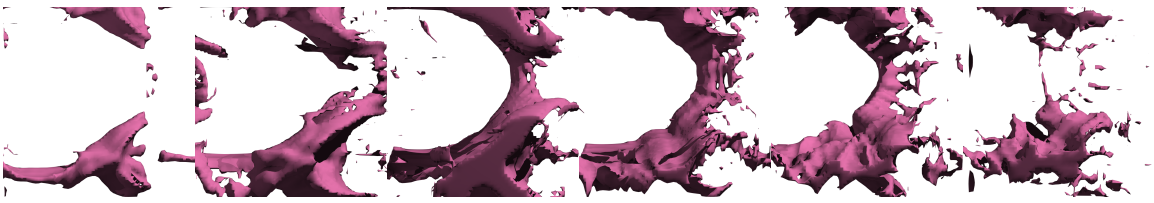


Figure 43: Some time slices of one isosurface of the fluid simulation data set.

Table 7: Topological complexity and running times for fluid simulation data set on a 2 GHz Pentium 4. Threshold value is 25 (11% of the height field range). The number of voxels is $64 \times 64 \times 32 \times 41 = 5,373,952$.

		From inside		From outside	
Topology check	Critical before	Critical after	Time (min)	Critical after	Time (min)
slices	722350	206433	8:38	206030	8:20
fully 3D	722350	549981	12:48	556495	11:22
combo	722350	227778	29:27	204934	11:30

the slice-by-slice carving order.

6.6 Conclusions

The continuous interpretation of the contour trees of the thick slices depends on a definition of the interpolation used on the values between the slices. Often linear interpolation is used, but as shown in Figure 1 of [85] it can cause artifacts in the interslice topology. Sohn and Bajaj [81] suggest a method of checking the amount of overlap between sublevel sets in adjacent time slices to avoid mistakes in tracking. The isosurface algorithm described in Section 3.3 gives an extension of a cubical scalar field to a continuous scalar field h as follows: For a point x, define $h(x)$ as the infimum of all isovalues whose isosurface has x on the inside. However, we propose that the method of interpolation does not matter much, in the sense that the resulting topological analyses are not very different. Therefore, a method of interpolation that reduces the size of the topological representation is more important. If the size is reduced, operations such as queries and contour tracking on the representation may be performed more efficiently.

Furthermore, if the inclusion-induced maps from the slice contour trees to the thick slice contour trees are 1-1 for a large percentage of the contour tree, then it will be easier to track the contours throughout time, since they are not merging or separating much over time. Therefore, one way to measure the benefit of topology

Table 8: Violations of 1-1 of the inclusion-induced map for the thick slices of the spiral wave data set. Threshold value is 10 (3.9% of the height field range).

Topology check	weighted	unweighted
slices	23291	29847
fully 3D	22304	23491
combo	19122	1772

simplification is to measure the decrease in violations of 1-1 for the inclusion-induced maps. Sohn and Bajaj [81] use the amount of overlap of sublevel set components to connect contours through time. They show an example of a case where without this the components merge in an undesirable way. Our approach measures the same problem using instead the 1-1 violations in the inclusion-induced maps. See Table 8 for a measure of the 1-1 violations after simplification of the spiral wave data set (Figure 41). The unweighted measure is simply the number of extra slice tree edges mapped to a thick slice tree edge plus the number of thick slice tree edges with no slice tree edges mapped to them. The weighted measure multiplies each counted edge by its length (difference in height value of its endpoints). See also the one dimension plus time example in Figure 44. If the 1-1 violations are reduced, then the contours will be tracked more effectively.

Therefore, it is beneficial to use topology simplification on parameter-dependent volume data for two reasons. First, it reduces the size of the thick slice contour trees to make queries more efficient. Second, it reduces the number of violations of 1-1 in the inclusion-induced maps.

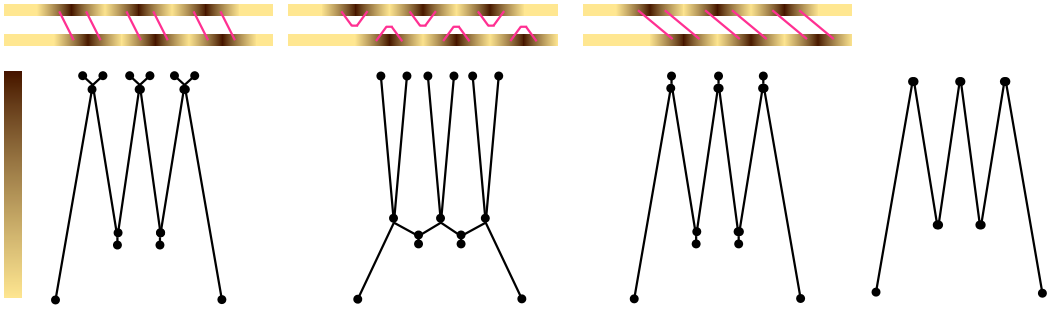


Figure 44: Contour trees of two different thick slices. The volume is three dark spots moving right. In the second figure the spots are moving faster. Note that the correspondence has been lost. The third figure is the contour tree of the thick slice after topology simplification (in both cases), and the fourth figure is the contour tree of any of the individual slices. Note that the inclusion-induced maps from the slice into the thick slice is much closer to 1-1 after simplification, especially in the second case.

CHAPTER 7

CONCLUSION

7.1 Summary

Global topological features are the result of local changes. This is the key to analyzing the topology of volumetric sets. In this thesis we have applied this to cubical scalar fields for the purposes of surface genus reduction, volume critical point reduction, and topological representation construction.

One of the contributions of this work is the use of different local topology checks for different purposes. The most general test checks for any change in topology. Under certain assumptions about the desired result (only one component), we can use a different test that checks only for increases in genus to get a cleaner result. Finally, for parameter-dependent data, we use a combination check that checks both for a topology change in the full volume and for a change in the parameter slice. Again the output is cleaner and more useful for tracking contours over changes in the parameter.

The next contribution is the use of a fast greedy algorithm for topology simplification with minimal user intervention. We have applied an isosurface topology simplification algorithm to surfaces, volumes, and parameter-dependent volumes, in two, three, and four dimensions with compelling results. The algorithm requires only one parameter to be set by the user—a measure of the importance of the desired topological features of the output.

Finally, we have also improved on the greedy algorithm for surfaces by applying it to an octree representation and to partially-defined volumes. Applying the multiresolution implementation to the octree results in sublinear (in the number of

voxels) time complexity for reasonable models. The parallel computation of an internal voxel set and an external voxel set creates a watertight extracted isosurface from partially-defined volume data.

7.2 Difficult examples

7.2.1 Suboptimal performance

There are cases where small perturbations of the input data can cause different numbers of critical points after running topology simplification. See Figure 45 for a 2D example. All the voxels left have the same value, and all will cause a topology change when carved. A slight perturbation of the data will determine which topology-altering voxel is carved first. Depending on which is carved first could result in two topology-altering carves or three topology-altering carves, and therefore two or three critical points (plus the global minimum) in the output volume. Since our algorithm makes all decisions locally and without backtracking, it cannot determine the optimal perturbation.

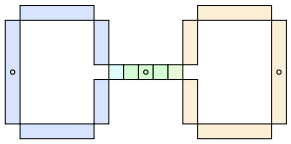


Figure 45: Possible suboptimal carving order in 2D. Carving a blue or yellow voxel will result in only two topology-altering carves, because once one of the rings opens up the green voxels will be carved without changing topology. However, starting with a green voxel (in the middle) will result in three topology-altering carves (plus the global minimum).

In our experience this suboptimal performance in practice is not an issue, because it is only a small difference in the number of critical points in the output. However, it would be interesting to describe an algorithm to produce the minimum number of critical points. One possibility is to do some global analysis to determine which topology-altering voxel to carve. In the example, there are three groups of voxels that are connected at two points. The correct voxel to carve would be from one of

the groups connected to only one other group. Proving optimality of this sort of procedure would be important. Another possible algorithm would be to try carving all possible topology-altering voxels, one at a time, and see which results in the fewest critical points. Such a brute force algorithm would not be practical, but it would be guaranteed to find the lowest number of critical points, as long as the intermediate voxel sets are collapsible to a single voxel (see next section).

Overall, the goal is to create an algorithm that simplifies the topology of a volume to the point described by the user while keeping the the result as close to the original as possible, in terms of number of voxels changed and how much they are changed.

7.2.2 Termination

With a finite threshold, our algorithm for volume simplification will always terminate, because eventually any voxel will be delayed up to the threshold and then there is no restriction on carving. However, if the user wishes to eliminate all critical points (except the global min or max), an infinite threshold can be specified. In such a case the algorithm does not allow any topology-altering operations. In 2D this does not cause any problems. The example above, for instance, can never be reached because topology-altering operations must have already been performed to get to this state. In 3D, however, there are situations where all remaining voxels cause a topology change and the state was reached from a topology-simple set. The simplest example we can think of where this happens is the "house of two rooms" [19]. See Figure 46. The problem is the existence of sets that cannot be collapsed to a point without first being expanded to another set. Since our algorithm consists of a series of expansions or a series of collapses, such sets cannot be avoided. We call them topological obstacles. In order to handle such cases the algorithm would have to follow a series of collapses and expansions, but there is no known simple way to do this in general [19]. In our experiments these topological obstacles have not been a problem in practice.

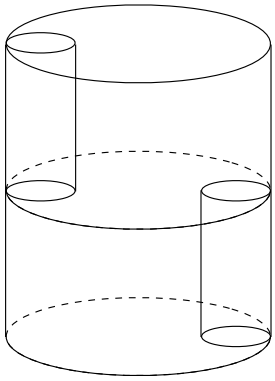


Figure 46: The "house of two rooms." You get to each room via the tunnel going through the other room. It is formed by expanding a point to a solid cylinder and collapsing the cylinder to this hollow shape. It cannot hence be collapsed to a point, however, so the collapsing version of our algorithm will get stuck here if this shape is ever formed.

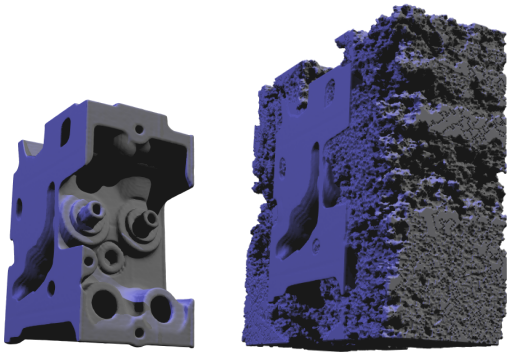


Figure 47: Result when carving in what is essentially a random order. Left is an isosurface from the input data, and right is the isosurface from the result of carving from outside.

We have seen this problem come up when extracting surfaces from volume data sets. The engine model (see Figure 47) has large nearly flat, but slightly noisy, regions. The result of carving based on the volume data constructs many topological obstacles. This is one reason we use the distance to the isosurface to control the carving order when just one isosurface is desired (Section 4.1).

7.3 Future Work

7.3.1 Applications

There are many applications of these techniques. Topology repair of laser range scans and tracking contours through slices are only a few. I would like to see this technique applied in even more computer graphics and medical visualization applications. As computer memory constraints weaken, more volumetric methods can be used. Programmable graphics cards may also be used in the future on volumetric data. It would also be interesting to see a parallel implementation of these techniques, since the algorithm is highly space-coherent.

Some algorithms for medical diagnosis require simple topology of the models being studied. Many organs (such as brains and spleens [67]) should be topologically equivalent to spheres, but CT and MRI scans of these organs are always noisy.

7.3.2 User guidance

The algorithms in this thesis have minimal user input, simply one value that controls the topological complexity of the output. However, more user input could be used to guide the carving procedure. For example, an interface could be built to allow the user to mark (perhaps roughly planar) regions of voxels as "uncarveable" so that the carving process would carve everything up to those regions and thereby create the patches at the regions specified by the user. This could assist in improving the suboptimal performance described above.

Furthermore, it would be nice to give the user control over what changes are made to the data. When we repair the topology of a surface, either all the handles are broken or all are patched. Small changes are not a problem, but for larger changes a mechanism and an interface need to be designed that allow users to decide on a per-handle basis what to do. The changes made to the shape of the volumes are also determined by our algorithm. In 2D, the changes result in leveled-out maxima and

canals between minima (or vice versa, see Figure 26 and Section 4.2). Carr's contour tree simplification approach [15] levels out peaks and fills in minima. Again it would be nice to allow the user to decide whether a maximum should be removed by leveling or a saddle point removed by constructing a ridge or canal.

Finally, in order to deal with the topological obstacles discussed above, a method of backtracking could be added to this procedure. This would almost certainly involve a priori knowledge of specific types of data, because checking all possible points of backtracking is prohibitive.

7.3.3 Theoretical underpinning

It will be interesting to see what insight can be gained from the description of scalar fields as a mathematical category. In particular, I would like to see descriptions of the relationship between contour trees and Morse-Smale complexes, especially in higher dimensions.

7.3.4 Extensions

Further extensions of this work could include topological analysis of other fields, such as vector fields. Also, as the dimensionality of the scalar field increases, the complexity of the local neighborhood increases. Strides should be taken in the direction of simplifying the local check in higher-dimensional cases. More work should be done along the lines of Bischoff and Kobbelt's approach [8] to see what can be achieved by representing the voxel set as a cell complex, since much of the trouble caused by complex critical points is eliminated.

Finally, I would like to see a multiple-frequency or multiple-pass approach to topology analysis that can be used to describe locations in the data of the topological features, such as handles of a surface, that are notoriously difficult to define. The problem we experienced with using a single-pass voxel carving algorithm is that small features get lost while detecting large features (see Figure 48). A better algorithm

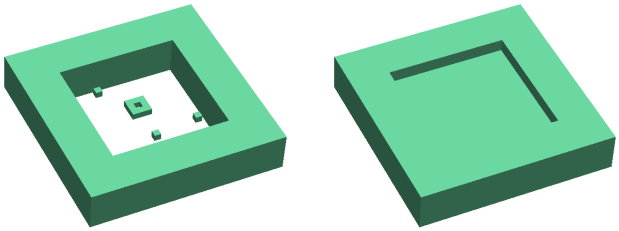


Figure 48: Topological feature detection. Our algorithms need to be extended for topological analysis. In this example, we can only detect the large handle because the carving procedure covers up the small topological features while detecting the large handle. Left is input, right is output.

might detect successively smaller features in successive passes. This would be very useful in a user-guided system for interactive editing of topological features.

REFERENCES

[1] Aktouf, Z., Bertrand, G., and Perroton, L., "A 3D-hole closing algorithm," in *Discrete Geometry for Computer Imagery*, vol. 1176 of *Lecture Notes in Computer Science*, pp. 36–47, Springer Berlin / Heidelberg, 1996.

[2] Aktouf, Z., Bertrand, G., and Perroton, L., "A 3D-hole closing algorithm," *Pattern Recognition Letters*, vol. 23, pp. 523–531, 2002.

[3] Ayala, R., Domínguez, E., Francés, A. R., and Quintero, A., "Homotopy in digital spaces," *Discrete Appl. Math.*, vol. 125, no. 1, pp. 3–24, 2003.

[4] Banks, D. C. and Linton, S., "Counting cases in marching cubes: Toward a generic algorithm for producing substitopes," in *VIS '03: Proceedings of the 14th IEEE Visualization 2003 (VIS'03)*, (Washington, DC, USA), p. 8, IEEE Computer Society, 2003.

[5] Berglund, N. and Szymczak, A., "Making contour trees subdomain-aware," in *Proceedings of the 16th Canadian Conference on Computational Geometry (CCCG'04)*, pp. 188–191, 2004.

[6] Bhaniramka, P., Wenger, R., and Crawfis, R., "Isosurfacing in higher dimensions," in *Proceedings of IEEE Visualization 2000* (Ertl, T., Hamann, B., and Varshney, A., eds.), pp. 267–273, 2000.

[7] Bhaniramka, P., Wenger, R., and Crawfis, R., "Isosurface construction in any dimension using convex hulls," *IEEE Transactions on Visualization and Computer Graphics*, vol. 10, no. 2, pp. 130–141, 2004.

[8] Bischoff, S. and Kobbelt, L., "Sub-voxel topology control for level-set surfaces," *Eurographics*, vol. 22, no. 3, 2003.

[9] Boissonnat, J.-D. and Cazals, F., "Smooth surface reconstruction via natural neighbour interpolation of distance functions," *Computational Geometry*, vol. 22, pp. 185–203, May 2006.

[10] Bottino, A., Nuij, W., and Van Overveld, C., "How to shrinkwrap through a critical point: an algorithm for the adaptive triangulation of isosurfaces with arbitrary topology," in *Proc. Implicit Surfaces*, pp. 53–72, October 1996.

[11] Bredno, J., Lehmann, T. M., and Spitzer, K., "A general discrete contour model in two, three, and four dimensions for topology-adaptive multichannel segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 25, no. 5, pp. 550–563, 2003.

[12] Bremner, P.-T., Edelsbrunner, H., Hamann, B., and Pascucci, V., "A multi-resolution data structure for two-dimensional morse-smale functions," in *Proc. IEEE Visualization 2003*, pp. 139–146, October 2003.

[13] Carr, H., *Topological Manipulation of Isosurfaces*. PhD thesis, University of British Columbia, 2004.

[14] Carr, H., Snoeyink, J., and Axen, U., "Computing contour trees in all dimensions," *Computational Geometry*, vol. 24, pp. 75–94, 2003.

[15] Carr, H., Snoeyink, J., and van de Panne, M., "Simplifying flexible isosurfaces using local geometric measures," *Proc. IEEE Visualization*, pp. 497–504, 2004.

[16] Chernyaev, E., "Marching cubes 33: Construction of topologically correct isosurfaces," Tech. Rep. CN 95-17, CERN, 1995.

[17] Chiang, Y. and Lu, X., "Progressive simplification of tetrahedral meshes preserving all isosurface topologies," 2003.

[18] Chiang, Y.-J., Lenz, T., Lu, X., and Rote, G., "Simple and optimal output-sensitive construction of contour trees using monotone paths," *Comput. Geom. Theory Appl.*, vol. 30, no. 2, pp. 165–195, 2005.

[19] Cohen, M. M., *A Course in Simple-Homotopy Theory*. Graduate Texts in Mathematics, New York: Springer-Verlag, 1970.

[20] Cohen-Steiner, D., Edelsbrunner, H., and Harer, J., "Stability of persistence diagrams," in *SCG '05: Proceedings of the twenty-first annual symposium on Computational geometry*, (New York, NY, USA), pp. 263–271, ACM Press, 2005.

[21] Cole-McLaughlin, K., Edelsbrunner, H., Harer, J., Natarajan, V., and Pascucci, V., "Loops in reeb graphs of 2-manifolds," in *SCG '03: Proceedings of the nineteenth annual symposium on Computational geometry*, (New York, NY, USA), pp. 344–350, ACM Press, 2003.

[22] Curless, B. and Levoy, M., "A volumetric method for building complex models from range images," in *Proc. of SIGGRAPH 1996*, pp. 4–9, August 1996.

[23] Davis, J., Marschner, S., Garr, M., and Levoy, M., "Filling holes in complex surfaces using volumetric diffusion," in *First International Symposium on 3D Data Processing, Visualization, and Transmission*, June 2002.

[24] Delfinado, C. J. A. and Edelsbrunner, H., "An incremental algorithm for betti numbers of simplicial complexes on the 3-sphere," *Computer Aided Geometric Design*, vol. 12, pp. 771–784, November 1995.

[25] Edelsbrunner, H., Harer, H., and Zomorodian, A., "Hierarchical morse-smale complexes for piecewise linear 2-manifolds," in *Symp. on Computational Geometry*, pp. 70–79, ACM, ACM Press, New York, NY, USA, 2001.

[26] Edelsbrunner, H., Harer, J., Natarajan, V., and Pascucci, V., "Morse-smale complexes for piecewise linear 3-manifolds," in *Proc. 19th Ann. Sympos. Comput. Geom.*, pp. 361–370, 2003.

[27] Edelsbrunner, H., Letscher, D., and Zomorodian, A., "Topological persistence and simplification," *Discrete and Computational Geometry*, vol. 28, no. 4, pp. 511–533, 2002.

[28] Edelsbrunner, H., Harer, J., Mascarenhas, A., and Pascucci, V., "Time-varying reeb graphs for continuous space-time data," in *SCG '04: Proceedings of the twentieth annual symposium on Computational geometry*, (New York, NY, USA), pp. 366–372, ACM Press, 2004.

[29] Edelsbrunner, H., Harer, J., Natarajan, V., and Pascucci, V., "Local and global comparison of continuous functions," in *IEEE Conference on Visualization*, pp. 275–280, 2004.

[30] Edelsbrunner, H. and Zomorodian, A., "Computing linking numbers of a filtration," *Homology, Homotopy and Applications*, vol. 5, no. 2, pp. 19–37, 2003.

[31] El-Sana, J. and Varshney, A., "Topology simplification for polygonal virtual environments," *IEEE Transactions on Visualization and Computer Graphics*, vol. 4, pp. 133–144, /1998.

[32] El-Sana, J. and Varshney, A., "Controlled simplification of genus for polygonal models," in *Proc. IEEE Visualization '97* (Yagel, R. and Hagen, H., eds.), pp. 403–412, 1997.

[33] Erikson, C., "Polygonal simplification: An overview," Tech. Rep. TR96-016, UNC Chapel Hill Computer Science, 1996.

[34] Fleishman, S., Drori, I., and Cohen-Or, D., "Bilateral mesh denoising," *ACM Trans. Graph.*, vol. 22, no. 3, pp. 950–953, 2003.

[35] Forman, R., "A user's guide to discrete morse theory."

[36] Forman, R., "Discrete Morse theory and the cohomology ring," *Trans. Amer. Math. Soc.*, vol. 354, pp. 5063–5085, 2002.

[37] Gansner, E. R., Koutsofios, E., North, S. C., and Vo, K.-P., "A technique for drawing directed graphs," *IEEE Transactions on Software Engineering*, vol. 19, no. 3, pp. 214–230, 1993.

[38] Gansner, E. R. and North, S. C., "An open graph visualization system and its applications to software engineering," *Software-Practice and Experience*, vol. 30, no. 11, pp. 1203–1233, 2000.

[39] Garland, M. and Heckbert, P. S., "Surface simplification using quadric error metrics," in *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, (New York, NY, USA), pp. 209–216, ACM Press/Addison-Wesley Publishing Co., 1997.

[40] Gau, C. J. and Kong, T. Y., "Minimal non-simple sets in 4D binary images," *Graphical Models*, vol. 65, pp. 112–130, 2003.

[41] Gerstner, T. and Pajarola, R., "Topology preserving and controlled topology simplifying multiresolution isosurface extraction," in *Proc. IEEE Visualization 2000* (Ertl, T., Hamann, B., and Varshney, A., eds.), pp. 259–266, 2000.

[42] Gumhold, S., Wang, X., and McLeod, R., "Feature extraction from point clouds," in *Proceedings, 10th International Meshing Roundtable* (Laboratories, S. N., ed.), pp. 293–305, October 2001.

[43] Guskov, I. and Wood, Z., "Topological noise removal," in *Proc. Graphics Interface 2001* (Watson, B. and Buchanan, J. W., eds.), pp. 19–26, 2001.

[44] Gyulassy, A., Natarajan, V., Pascucci, V., Bremer, P.-T., and Hamann, B., "Topology-based simplification for feature extraction from 3D scalar fields," in *IEEE Visualization*, p. 68, IEEE Computer Society, 2005.

[45] Gyulassy, A., Natarajan, V., Pascucci, V., Bremer, P.-T., and Hamann, B., "A topological approach to simplification of three-dimensional scalar functions," *IEEE Transactions on Visualization and Computer Graphics*, vol. 12, no. 4, pp. 474–484, 2006.

[46] Han, X., Xu, C., and Prince, J. L., "A topology preserving deformable model using level sets," in *Proc. IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR2001)*, pp. 765–770, December 2001.

[47] Han, X., Xu, C., and Prince, J. L., "Topology preserving geometric deformable models for brain reconstruction," in *Geometric Level Set Methods in Imaging, Vision and Graphics* (Osher, S. and Paragios, N., eds.), pp. 421–438, Springer-Verlag, 2003.

[48] Han, X., Xu, C., and Prince, J. L., "A topology preserving level set method for geometric deformable models.," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 25, no. 6, pp. 755–768, 2003.

[49] He, T., Hong, L., Varshney, A., and Wang, S. W., "Controlled topology simplification," *IEEE Transactions on Visualization and Computer Graphics*, vol. 2, no. 2, pp. 171–184, 1996.

[50] Hilaga, M., Shinagawa, Y., Kohmura, T., and Kunii, T. L., "Topology matching for fully automatic similarity estimation of 3D shapes," in *SIGGRAPH*

'01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques, (New York, NY, USA), pp. 203–212, ACM Press, 2001.

[51] Hoppe, H., DeRose, T., Duchamp, T., McDonald, J., and Stuetzle, W., "Mesh optimization," in *Proc. of SIGGRAPH 1993*, pp. 19–26, 1993.

[52] Hoppe, H., DeRose, T., Duchamp, T., McDonald, J., and Stuetzle, W., "Surface reconstruction from unorganized points," in *SIGGRAPH '92: Proceedings of the 19th annual conference on Computer graphics and interactive techniques*, (New York, NY, USA), pp. 71–78, ACM Press, 1992.

[53] Joswig, M. and Pfetsch, M. E., "Computing optimal Morse matchings." http://front.math.ucdavis.edu/math.CO/0408331.

[54] Joswig, M. and Pfetsch, M. E., "Computing optimal morse matchings," *SIAM J. Discret. Math.*, vol. 20, no. 1, pp. 11–25, 2006.

[55] Ju, T., Zhou, Q.-Y., and Hu, S.-M., "Editing the topology of 3D models by sketching," in *Proc. of SIGGRAPH 2007*, 2007.

[56] Ju, T., "Robust repair of polygonal models," *ACM Trans. Graph.*, vol. 23, no. 3, pp. 888–895, 2004.

[57] Kaczinski, T., Mischaikow, K., and Mrozek, M., *Computational homology*. New York, NY: Springer-Verlag, 2004.

[58] Kaczynski, T., Mischaikow, K., and Mrozek, M., "Computing homology," *Homology, Homotopy, and Applications*, vol. 5, no. 2, pp. 233–256, 2003.

[59] Kazhiyur-Mannar, R., Wenger, R., Crawfis, R., and Dey, T. K., "Adaptive resolution isosurface construction in three and four dimensions," Tech. Rep. OSU-CISRC-7/03-TR38, The Ohio State University, 2004.

[60] Kim, B., Liu, Y., Llamas, I., and Rossignac, J., "Advections with significantly reduced dissipation and diffusion," *IEEE Transactions on Visualization and Computer Graphics*, vol. 13, no. 1, pp. 135–144, 2007.

[61] Lee, H., Desbrun, M., and Schröder, P., "Progressive encoding of complex isosurfaces," *ACM Trans. Graph.*, vol. 22, no. 3, pp. 471–476, 2003.

[62] Levoy, M., Pulli, K., Curless, B., Rusinkiewicz, S., Koller, D., Pereira, L., Ginzton, M., Anderson, S., Davis, J., Ginsberg, J., Shade, J., and Fulk, D., "The digital Michelangelo project: 3D scanning of large statues," in *Proc. of SIGGRAPH 2000*, pp. 131–144, 2000.

[63] Lewiner, T., Lopes, H., and Tavares, G., "Applications of Forman's discrete Morse theory to topology visualization and mesh compression," *IEEE Transactions on Visualization and Computer Graphics*, vol. 10, no. 5, pp. 499–508, 2004.

[64] Lorensen, W. E. and Cline, H. E., "Marching cubes: A high resolution 3D surface reconstruction algorithm," *ACM Computer Graphics*, vol. 21, no. 3, pp. 163–169, 1987.

[65] Ma, K.-L., "Visualizing time-varying volume data," *Computing in Science and Engg.*, vol. 5, no. 2, pp. 34–42, 2003.

[66] Mirin, A., Cohen, R., Curtis, B., Dannevik, W., Dimitis, A., Duchaineau, M., Eliason, D., Schikore, D., Anderson, S., Porter, D., Woodward, P., Shieh, L., and White, S., "Very high resolution simulation of compressible turbulence on the ibmsp system," in *Supercomputing '99*, 1999.

[67] Nain, D., Haker, S., Bobick, A., and Tannenbaum, A., "Shape-driven 3d segmentation using spherical wavelets," in *Medical Image Computing and Computer-Assisted Intervention: MICCAI 2006*, pp. 66–74, 2006.

[68] Niethammer, M., Kalies, W. D., Mischaikow, K., and Tannenbaum, A., "On the detection of simple points in higher dimensions using cubical homology," *IEEE Transactions on Image Processing*, vol. 15, no. 8, pp. 2462–2469, 2006.

[69] Nooruddin, F. S. and Turk, G., "Simplification and repair of polygonal models using volumetric techniques," *IEEE Transactions on Visualization and Computer Graphics*, vol. 9, no. 2, pp. 191–205, 2003.

[70] Pajarola, R. and Gerstner, T., "Topology control in multiresolution isosurface extraction," Tech. Rep. UCI-ICS-05-01, University of California Irvine, 2005.

[71] Pascucci, V. and Cole-McLaughlin, K., "Efficient computation of the topology of level sets," *Proc. IEEE Visualization*, 2002.

[72] Pascucci, V., Cole-McLaughlin, K., and Scorzelli, G., "Multiresolution computation and presentation of contour trees," Tech. Rep. UCRL-PROC-208680, Lawrence Livermore National Laboratory, 2004.

[73] Pascucci, V. and Cole-McLaughlin, K., "Parallel computation of the topology of level sets," *Algorithmica*, vol. 38, no. 1, pp. 249–268, 2003.

[74] Rana, S., *Topological Data Structures for Surfaces*. Chichester, West Sussex, England: John Wiley & Sons, Ltd, 2004.

[75] Samet, H., *Applications of Spatial Data Structures*. Reading, MA: Addison-Wesley, 1990.

[76] Sander, P., Snyder, J., Gortler, S., and Hoppe, H., "Texture mapping progressive meshes," in *Proc. of SIGGRAPH 2001*, pp. 409–416, 2001.

[77] Sharf, A., Alexa, M., and Cohen-Or, D., "Context-based surface completion," in *Proc. of SIGGRAPH 2004*, pp. 878–887, 2004.

[78] Sharf, A., Lewiner, T., Shamir, A., Kobbelt, L., and Cohen-Or, D., "Competing fronts for coarse-to-fine surface reconstruction," *Computer Graphics Forum*, vol. 25, no. 3, pp. 389–398, 2006.

[79] Sharf, A., Lewiner, T., Shklarski, G., Toledo, S., and CohenOr, D., "Interactive topology-aware surface reconstruction," in *Proc. of SIGGRAPH 2007*, 2007.

[80] Shinagawa, Y. and Kunii, T. L., "Constructing a reeb graph automatically from cross sections," *IEEE Comput. Graph. Appl.*, vol. 11, no. 6, pp. 44–51, 1991.

[81] Sohn, B.-S. and Bajaj, C., "Time-varying contour topology," *IEEE Transactions on Visualization and Computer Graphics*, vol. 12, no. 1, pp. 14–25, 2006.

[82] Steen, L. A. and Seebach, J. A., *Counterexamples in Topology*. Dover Publications, 1995.

[83] Steinke, F., Schlkopf, B., and Blanz, V., "Support vector machines for 3D shape processing," *Eurographics*, vol. 24, no. 3, pp. 285–294, 2005.

[84] Szymczak, A. and Vanderhyde, J., "Extraction of topologically simple isosurfaces from volume datasets," in *Proc. IEEE Visualization 2003*, pp. 67–74, October 2003.

[85] Szymczak, A., "Subdomain aware contour trees and contour evolution in time-dependent scalar fields," in *SMI '05: Proceedings of the International Conference on Shape Modeling and Applications*, (Washington, DC, USA), pp. 136–144, IEEE Computer Society, 2005.

[86] Szymczak, A., Tannenbaum, A., and Mischaikow, K., "Coronary vessel cores from 3D imagery: a topological approach," *Medical Imaging 2005: Image Processing*, vol. 5747, no. 1, pp. 505–513, 2005.

[87] Szymczak, A. and Vanderhyde, J., "Simplifying the topology of volume datasets: an opportunistic approach," *ACM Transactions on Graphics*, under review.

[88] Takahashi, S., Fujishiro, I., and Takeshima, Y., "Interval volume decomposer: A topological approach to volume traversal," in *Proceedings of Visualization and Data Analysis 2005*, pp. 103–114, January 2005.

[89] van Kreveld, M. J., van Oostrum, R., Bajaj, C. L., Pascucci, V., and Schikore, D., "Contour trees and small seed sets for isosurface traversal," in *Symposium on Computational Geometry*, pp. 212–220, 1997.

[90] Van Overveld, C. and Wyvill, B., "Shrinkwrap: an adaptive algorithm for polygonizing an implicit surface," Tech. Rep. 93/514/19, Department of Computer Science, University of Calgary, 1993.

[91] Vanderhyde, J. and Szymczak, A., "Topological simplification of isosurfaces in volumetric data using octrees," *Graphical Models*, to appear.

[92] Vieira, A. W., Velho, L., Lopes, H., Tavares, G., and Lewiner, T., "Fast stellar mesh simplification," in *Computer Graphics and Image Processing, 2003. SIBGRAPI 2003.*, pp. 27–34, IEEE Press, 2003.

[93] Wood, Z., Hoppe, H., Desbrun, M., and Schröder, P., "Isosurface topology simplification," Tech. Rep. MSR-TR-2002-28, Microsoft Research, January 2002.

[94] Wood, Z., Hoppe, H., Desbrun, M., and Schröder, P., "Removing excess topology from isosurfaces," *ACM Transactions on Graphics*, vol. 23, pp. 190–208, April 2004.

[95] Wood, Z. J., Schröder, P., Breen, D., and Desbrun, M., "Semi-regular mesh extraction from volumes," in *VIS '00: Proceedings of the conference on Visualization '00*, (Los Alamitos, CA, USA), pp. 275–282, IEEE Computer Society Press, 2000.

[96] Zhang, E., Mischaikow, K., and Turk, G., "Feature-based surface parameterization and texture mapping," *ACM Transactions on Graphics*, vol. 24, pp. 1–27, January 2005.

[97] Zhang, N. and Kaufman, A., "Multiresolution volume simplification and polygonization," in *VG '03: Proceedings of the 2003 Eurographics/IEEE TVCG Workshop on Volume graphics*, (New York, NY, USA), pp. 87–94, ACM Press, 2003.

[98] Zhang, X., Bajaj, C., and Baker, N., "Fast matching of volumetric functions using multi-resolution dual contour trees," 2004.

[99] Zhao, H.-K., Osher, S., and Fedkiw, R., "Fast surface reconstruction using the level set method," in *VLSM '01: Proceedings of the IEEE Workshop on Variational and Level Set Methods (VLSM'01)*, (Washington, DC, USA), p. 194, IEEE Computer Society, 2001.

VITA

James N. Vanderhyde was born and raised in western Michigan, just outside of Grand Rapids. He graduated third in his class from Rockford High School in 1999 and attended Hope College as a National Merit finalist. He studied math, computer science, and German, and from then on he sought to combine studies in math and computer science. At Michigan State University he worked with Eric Torng and Charles Ofria in computational biology and earned his master's of science degree. In 2001 he started his PhD at Georgia Tech in Algorithms, Combinatorics, and Optimization, but he switched after two years to computer graphics. In the summer of 2007 he finished his PhD under the supervision of Dr. Andrzej Szymczak. James now lives with his wife Mariam in Atlanta, Georgia.